WebSphere® Business Integrator

Process Broker Services Developer's Guide

Version 2.1

WebSphere® Business Integrator

Process Broker Services Developer's Guide

Version 2.1

Note
Before using this information and the products it supports, read the information in "Notices" on page 103

Second Edition (December 2001)

This edition applies to Version 2.1 of the IBM® WebSphere® Business Integrator (program number 5724-A78) and to all subsequent release and modifications until otherwise indicated in new editions.

Contents

Figures

About this book

The *WebSphere Business Integrator Process Broker Services Developer's Guide* provides the information necessary to create and deploy Process Broker Services solution artifacts.

Who should read this book

This book is for solution developers who build and deploy the Process Broker Services solution artifacts. This book is also intended for those who perform any required initial problem determination.

What you need to know

Solution developers must have a solid understanding of Business Integrator Process Broker Services; the *WebSphere Business Integrator Concepts and Planning* book and the *WebSphere Business Integrator Process Broker Services Concepts Guide* provide the required background information.

Before you implement WebSphere Business Integrator Process Broker Services

WebSphere Business Integrator uses multiple underlying products and technologies to support the solutions that you create and run. In general, before you implement Business Integrator, you will need to understand the underlying products and technologies that support your solution.

Before you implement Business Integrator, you or other members of your organization will need to be generally skilled in the activities listed below for similar solutions, products and underlying products and technologies. If you and other members of your organization do not possess these skills, you will need to obtain assistance, from qualified services staff, either from IBM or from third parties, to implement Business Integrator. You must be prepared to use the documentation of the underlying products and technologies. (This documentation is provided with Business Integrator or otherwise from IBM.)

When you plan, install, and configure Business Integrator, you will need to understand how to install and configure some of the underlying products and technologies that you use in your installation. Business Integrator provides the installation of most of the underlying products and technologies into its run time environment. However, you might need to install and configure certain underlying products separately into either the build time or run time environment. You might also need to diagnose and correct installation problems with underlying products and technologies.

Before you design, develop and publish solutions, you will need to be:

- Generally familiar with system integration techniques in a business environment.
- Prepared to use the tools of the underlying products and technologies that your solution requires.
- Familiar with the run time behavior of the underlying products and technologies that your solution requires.
- Familiar with modeling concepts and techniques such as Unified Modeling Language, and related tools, with state machine concepts, and with visual flow composition-modeling concepts and techniques.
- Familiar with Internet and Electronic Data Interchange (EDI) concepts and technologies, if required by your solution.
- Prepared to research the existing applications, systems, and networks that you integrate with Business Integrator.
  - Inside your enterprise, they can be known as legacy systems, back-end systems, enterprise applications, or endpoint applications.
  - Outside your enterprise, they can be known as trading networks, private EDI networks, or similar networks that your solution requires.

Before you deploy, run, manage, diagnose, and tune Business Integrator, you will need to be prepared to use the management, trace, audit, exception handling, diagnostic and related tools of the underlying products and technologies that support your solution. You will need to be prepared to understand the solution itself to the degree needed for these tasks.

Conventions and terminology used in this book

The "Glossary of Terms and Abbreviations" on page 107 introduces the terminology relevant to Business Integrator Process Broker Services.

How to send your comments

IBM welcomes your comments. You can send your comments by any one of the following methods:

1. Electronically to this address:

   `idrcf@hursley.ibm.com`

   Be sure to include your network address if you want a reply.
2. By FAX, to the following numbers:

   `UK: 01962-842327`
   `Other countries: +44-1962-842327`
3. By mail to the following address:

User Technologies
Mail Point 095
IBM United Kingdom Laboratories
Hursley Park
Winchester
Hampshire
SO21 2JN
United Kingdom

Chapter 1. Introduction

Process Broker Services is the core of the Business Flow Manager component of WebSphere Business Integrator. This document is intended for solution developers who will build and deploy the Process Broker Services solution artifacts.

This guide provides the following information:

- "Chapter 2. Building and deploying solution artifacts" on page 3. This chapter describes how to build an adaptive document, and how to create and configure controllers.
- "Chapter 3. Client programming" on page 33. This chapter describes Web clients and messaging clients. It also provides information about handling automatic activities in workflows.
- "Chapter 4. Advanced Topics" on page 47. This chapter provides the following topics:
  - Life cycle management of adaptive documents (archiving, revival, and removal)
  - Process Broker Services Scheduler
  - Assigning an adaptive document identifier
  - Dynamically incorporating changes to controllers
  - Receiver caching
  - Tips
- "Chapter 5. Troubleshooting" on page 61. This chapter provides a checklist to use when troubleshooting. It also provides information about trace files and trace dumps.
- "Appendix A. Process Broker Services properties" on page 65. This appendix describes package names, file locations, trace and debug flags, scheduler dispatcher properties, and automated activities.
- "Appendix B. Process Broker Services Application Program Interfaces" on page 69. This appendix provides a summary of the interfaces, classes and methods.

Introduction

Chapter 2. Building and deploying solution artifacts

The solution artifacts for Process Broker Services include the adaptive documents and the associated controllers that define the dynamic behavior of adaptive documents. See the *WebSphere Business Integrator Process Broker Services Concepts Guide* for a functional overview of Process Broker Services and how adaptive documents enable collaborative business process management.

Building an adaptive document

Adaptive documents are entity beans with container- managed persistence; for

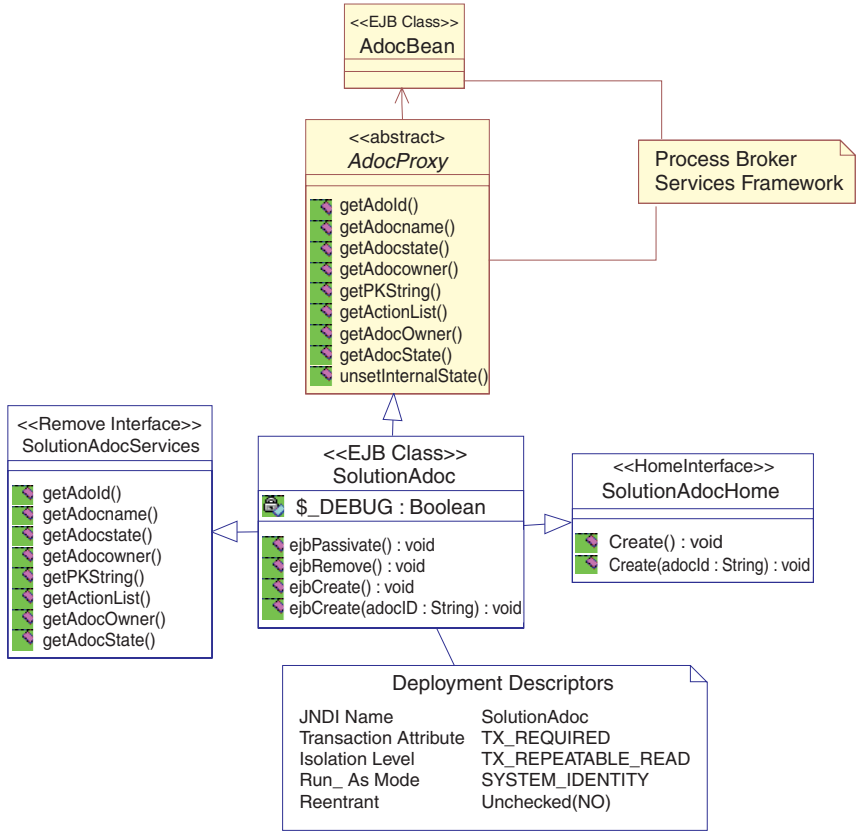

Figure 1. Solution adaptive document

example, the attributes of an adaptive document are persisted by the Enterprise Bean container. The Process Broker Services framework provides the base AdocBean and an abstract AdocProxy class that are accessed by importing `com.ibm.epic.bfm.ejb.base.*`. Building a solution specific adaptive document requires extending the AdocProxy class implementing the abstract methods of AdocProxy. See Figure 1.

Note: The convention for naming a solution adaptive document is *Solution*Adoc, where the word *Solution* can be replaced with any business entity in the solution, such as POAdoc for a Purchase Order adaptive document and so on.

The specific steps in building a SolutionAdoc are described in the following sections.

Creating the Enterprise Bean

1. In the VisualAge Java IDE, select the **EJB** tab. (If this tab is not there, select **File → Quick Start** and then **Features → Add Features** and add the Enterprise Bean Development environment). Add the Enterprise Bean Group for the solution, named as SampleApp as shown in Figure 2, and also define the project for the solution. All adaptive documents for the solution are now under this Enterprise Bean Group.

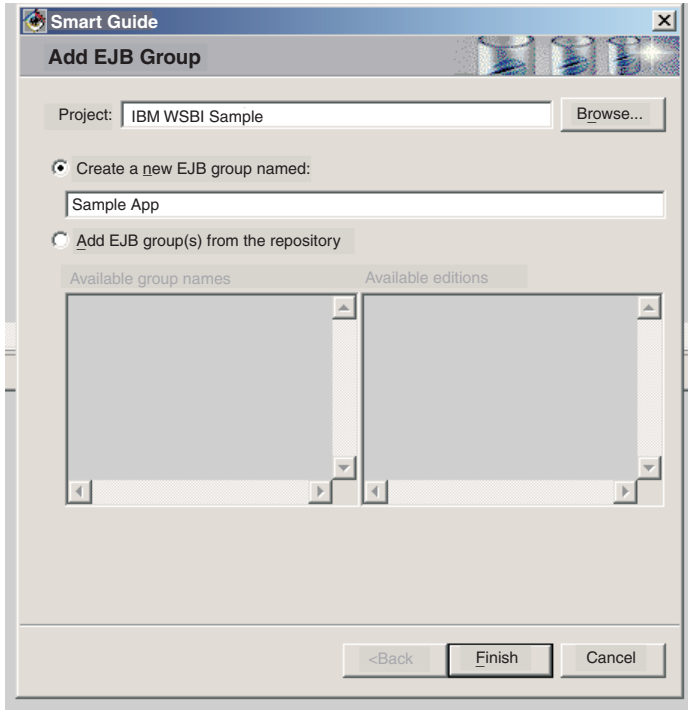

Figure 2. VisualAge Java -- Creating an Enterprise Bean Group

2. Create the SolutionAdoc Enterprise Bean under the SampleApp Enterprise Bean Group. The SolutionAdoc Bean has the import statement: `import com.ibm.epic.bfm.ejb.base.*`. The SolutionAdoc Enterprise Bean also has a variable *_DEBUG* that is initialized as follows:

```
private static boolean _DEBUG =
        BFMResources.getSingleton().getDebugInfo ("SolutionAdocDebug");
```

3. The SolutionAdoc extends the AdocProxy – `com.ibm.epic.bfm.base.AdocProxy`. VisualAge Java IDE might report

some errors at this stage, but the errors are resolved as the next few steps are performed.

Programming the home interface

4. The SolutionAdoc has a primary key that is `java.lang.string` and this key is initialized in the Create () method. The solutionId is assumed to be the primary key of the SolutionAdoc.
5. The SolutionAdoc Home Interface has the Create method with no arguments (see Figure 1 on page 4). The SolutionAdoc Bean Class has the corresponding ejbCreate method that is shown below.

```
public void ejbCreate() throws javax.ejb.CreateException,
                                java.rmi.RemoteException
 {

  try {
     // Intialize the base adoc
        solutionAdocid =super.createAdoc("Solution")

     // Initialize all the other fields of the ADOC here
     // ... ...

        if (_DEBUG)
         System.out.println ("SampleApp::SolutionAdoc created ..");
       }

    catch (Throwable e) {
          throw new RemoteException(e.getMessage()); }
  }
```

6. The SolutionAdoc Home Interface also has a Create method with the adocId as the argument (see Figure 1 on page 4). In this case, the ejbCreate method is used as shown in the next sample.

```
public void ejbCreate(java.lang.string argAdocId)
         throws javax.ejb.CreateException, java.rmi.RemoteException
        {
        _initLinks();

        // All Container Managed Persistence fields are initialized here

        try {
            adocId = argAdocId;
            if (_DEBUG)
            System.out.println ("SampleApp::SolutionAdoc.ejbCreate() –
            Creating SolutionAdoc with id " + adocId + ". Creating the Base ADOC");

       // Create the base ADOC
            super.createAdoc (adocId, "Solution");

            if (_DEBUG)
             System.out.println ("SampleApp::SolutionAdoc created ..");

        }
```

```
        catch (Throwable e) {
                        throw new RemoteException(e.getMessage()); }
}
```

Programming the remote interface

7. Since the SolutionAdoc extends the AdocProxy it implements the abstract methods specified in the AdocProxy class (see Figure 1 on page 4). Except for the unsetInternalState method, all the other methods are featured in the SolutionAdoc Remote Interface. The methods on the AdocProxy include:

   public string getAdocId ()
   throws RemoteException. See the get method pattern in step 8 on page 8 for the implementation.

   public string getAdocName ()
   throws RemoteException. The adaptive document name is also referred to as the adaptive document type. See the get method pattern in step 8 on page 8 for the implementation.

   public string getAdocOwner ()
   throws RemoteException. The adaptive document owner is the user who created the adaptive document. See the get method pattern in step 8 on page 8 for the implementation.

   public string getAdocState ()
   throws RemoteException. The adaptive document state is the current state of the document controller associated with the adaptive document. See the get method pattern in step 8 on page 8 for the implementation.

   public string getPKString ()
   This method establishes the link between the base adaptive document and the SolutionAdoc by having the same primary key for the base adaptive document (provided by the Process Broker Services framework) and the SolutionAdoc (see Figure 1 on page 4). The method returns the primary key.

   ```
   public string getPKString ( ) { return solutionId;}
   ```

   public java.util.vector getActionList ()
   throws RemoteException This method returns the events that can be raised on a given adaptive document instance based on the current state of the adaptive document controller.

   public void setAdocOwner (java.lang.string arg1)
   throws RemoteException. See the set method pattern in step 9 on page 8 for the implementation.

public void unsetInternalState ()
throws RemoteException; This is the only method on the AdocProxy class that is not promoted to the remote interface of the SolutionAdoc.

```
public void unsetInternalState () throws RemoteException
 {
       resetProxy();
       // any additional logic
 }
```

8. The Get Method Pattern for the get methods on the AdocProxy class – this is needed in order to avoid Enterprise Bean Inheritance using the AdocProxy. Enterprise Bean Inheritance is supported by VisualAge Java, which at the current time is not portable across other Enterprise Bean Containers.

```
public string getMethod ( ) throws Remote Exception
 {

  try {
       return getProxy().getMethod ();
     }

  catch (Throwable e) {
       throw new RemoteException
       ("SampleApp::SolutionAdoc.getMethod – " e.getMessage());}
  }
```

9. The Set Method Pattern for the set methods on the AdocProxy class – this is needed in order to avoid Enterprise Bean Inheritance using the AdocProxy. Enterprise Bean Inheritance is supported by VisualAge Java, which at the current time is not portable across other Enterprise Bean Containers.

```
public string setMethod (.... ) throws Remote Exception
 {

  try {
       return getProxy().setMethod (....);
     }

  catch (Throwable e) {
       throw new RemoteException
        ("SampleApp::SolutionAdoc.setMethod – " e.getMessage());}
 }
```

Additional methods on the solution adaptive document

10. In addition to the methods on the AdocProxy class, the SolutionAdoc Enterprise Bean must also implement the ejbPassivate and the ejbRemove methods.

```
public void ejbPassivate () throws RemoteException
  {
```

```
    this.unsetInternalState();
    // Any additional logic
  }

public void ejbRemove () throws
            java.rmi.RemoteException, javax.ejb.RemoveException
  {

    try {
           getProxy().remove();
        } catch (RemoveException re)
          {
           throw re;
          } catch (Throwable t)
           {

            if (_DEBUG)
            System.out.println("SampleApp::SolutionAdoc:
            ejbRemove() - " + t.getMessage());
            throw new RemoteException(
                 "SampleApp::SolutionAdoc:ejbRemove() –"
                        + t.getMessage());
           }
  }
```

Schema mapping for SolutionAdoc

The schema and data mapping for SolutionAdoc must follow the procedures as outlined in the VisualAge Java documentation. As mentioned before, the SolutionAdoc is an entity bean with container managed persistence. The solutionId is a mandatory attribute for the SolutionAdoc and it must be mapped to a VARCHAR of length 32. There is no restriction on the database (data source) used for the adaptive documents. However, the recommendation is to use the database used by the Business Flow Manager application server.

Deployment and configuration

The SolutionAdoc must have the following deployment descriptors:
JNDI Name – SolutionAdoc (Not the fully qualified name in VisualAge Java.)
Transaction Attribute – TX_REQUIRED
Isolation Level – T-REPEATABLE_READ
Run-As Mode – SYSTEM_IDENTITY
Reentrant – Unchecked(NO)

Additionally, all the get functions should be marked Read-Only. The following methods by default are marked Read-Only:
- archive
- getAdocId
- getAdocState

- getAdocOwner
- getAdocName
- getActionList.

For purposes of the configuration it is assumed that all the Solution adaptive documents are deployed in the same application server as Process Broker Services (a mandatory requirement in WebSphere Business Integrator Version 2.1). Process Broker Services has to know the package name of the SolutionAdoc Bean and this is done through the bfm.properties file – the following entry is made in the file: `SolutionAdocPackage = com.ibm.epic.solution.SampleApp`

Creating and configuring controllers

Controllers provide the dynamic behavior for the adaptive documents. There are two types of controllers in Process Broker Services: the Adaptive Document Controllers and the Activity Controllers (see Figure 3). See the *WebSphere Business Integrator Process Broker Services Concepts Guide* for an explanation of the controller concepts.

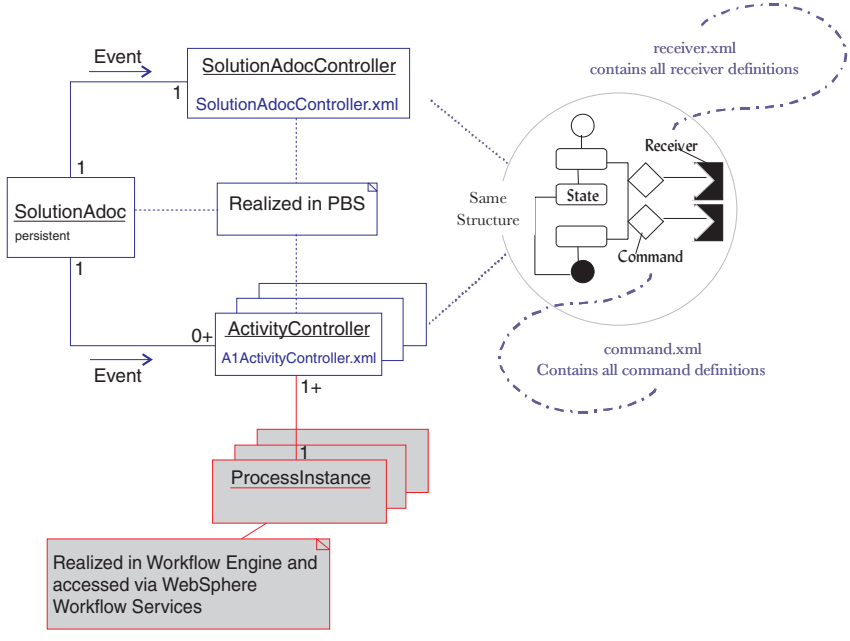

Figure 3. Adaptive document and activity controllers

The creation and configuration of the controllers involve the following steps:

1. **Define the adaptive document Controller**

   Each adaptive document has one controller; for example, in Figure 3 on page 10 the SolutionAdocController.xml defines the SolutionAdocController associated with the SolutionAdoc. The Adaptive Document Controller persists over the life-cycle of the adaptive document.

   The convention for naming the Adaptive Document Controller is based on the naming convention of the adaptive document. For example, the Adaptive Document Controller is named as SolutionAdocController, where the prefix Solution can be replaced by the solution entity. For example, for an RFQAdoc the Adaptive Document Controller is named as the RFQAdocController.

   See "Controller definition" on page 14 for details on the controller XML tags.

2. **Define the Activity Controllers**

   Multiple (zero or more) Activity Controllers can be associated with an adaptive document. An Activity Controller corresponds to an activity, in a given Process Instance, in which the adaptive document participates. The Activity Controller persists over the life-cycle of an activity and coordinates the tasks associated with the activity. Therefore, the set of Activity Controllers associated with an adaptive document changes over time as some activities are completed and new activities become available. Each activity controller definition is captured in an XML file. For example, in Figure 3 on page 10 the A1ActivityController.xml file defines the activity controller for an activity named A1.

   The convention for naming the Activity Controller is ActivityNameActivityController, where the ActivityName is replaced with the actual name of the activity.

   See "Controller definition" on page 14 for details on the controller XML tags.

3. **Associating the Controllers with the adaptive document**

   The adaptive document and the Activity Controllers are registered with Process Broker Services through entries in the bfm.properties file. For example, the Adaptive Document Controller is registered as:

   ```
   SolutionAdocControllerXML =
            \\WebSphere\\AppServer\\properties\\bfm\\SolutionAdocController.xml
   ```

   The Activity Controllers are similarly registered in the bfm.properties file as:

   ```
   A1ActivityControllerXML =
             \\WebSphere\\AppServer\\properties\\bfm\\A1ActivityController.xml
   ```

If the Activity Controller is not defined, then Process Broker Services uses a default activity controller for the activity. The default activity controller is named defaultActivityController and is registered in the bfm.properties file. See "Activity controllers" on page 16 for details on the defaultActivityController.

Once an Adaptive Document Controller is registered in the bfm.properties file using the naming convention as described in step 2 on page 11, the controller is associated with the named adaptive document. Similarly, an activity gets associated with the Activity Controller once it is registered in the bfm.properties file. The association of Activity Controllers to the adaptive document is dynamic and is handled by the Activity-Adaptive Document Map that is part of the Process Broker Services framework. Explicit user configuration is not required (see the *WebSphere Business Integrator Process Broker Services Concepts Guide* for an overview of the Activity-Adaptive Document).

4. **Implementing the Commands and Receivers**

   Commands are invoked from within transitions in the adaptive document and Activity Controllers. All the commands used in the various controllers are defined in a single command.xml file within Process Broker Services. The command.xml file is registered in the LDAP directory as shown in Figure 4 on page 13; if not, then it must be registered in the bfm.properties file. The commands represent the interface to business services provided by various end points and business objects. Each command has a unique identifier, the command id. More details on the XML tags for defining commands as well as the system commands that are predefined in the default command.xml file are provided in "Command definition" on page 19.

   Receivers can be viewed as implementations for the commands. The Flow Composition Builder is used to construct the receivers for the commands (See the *WebSphere Studio Business Integrator Extensions Installation Guide* for details on the Flow Composition Builder). All receivers used in Process Broker Services are defined in a single XML file within the Process Broker Services file receiver.xml. The receiver.xml file is registered in the LDAP directory as shown in Figure 4 on page 13, or alternately in the bfm.properties file. Each receiver has a unique identifier, the receiver id. The receivers are referenced from within the command definitions based on the receiver id. More details on the XML tags for defining receivers as well as the default receivers supplied with the Process Broker Services are described in "Receiver definitions" on page 28.

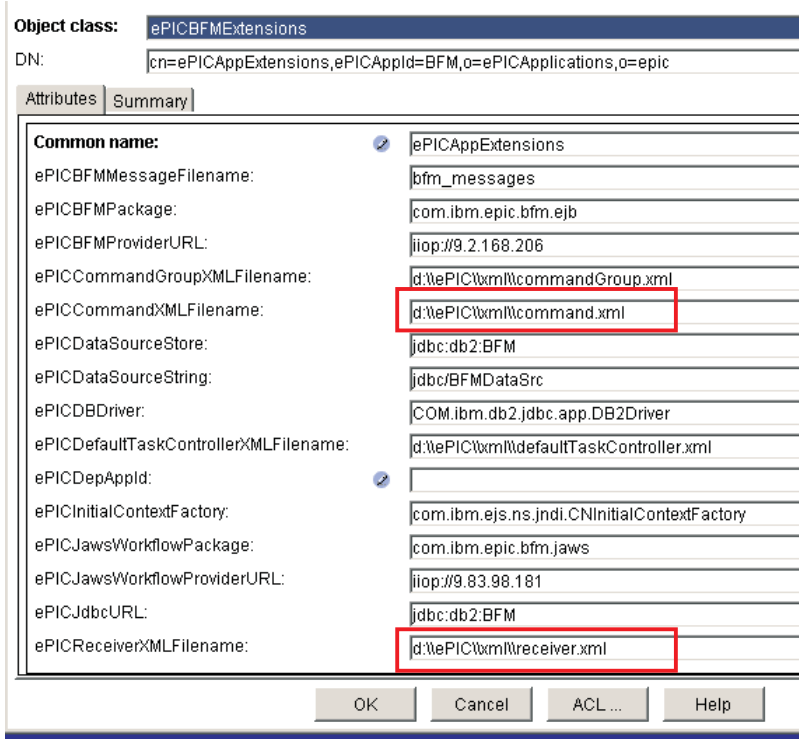

Figure 4. Registration of lists of Commands and Receivers in LDAP

Controller definition

The Adaptive Documents controllers and the Activity controllers, are defined in XML in accordance with the controller dtd (the controller XML Data Type Definition). The controller.dtd description follows:

```
<xml encoding="US-ASCII"?>
<!ELEMENT statemachine (name, directive?,state+)>
<!ATTLIST statemachine id ID #REQUIRED>
<!ELEMENT state (name,type?, transition*)>
<!ELEMENT name (#PCDATA)>
<!ELEMENT type (#PCDATA)>
<!ELEMENT transition (target, event?, condition?, directive?, action*)>
<!ELEMENT target (#PCDATA)>
<!ELEMENT event (#PCDATA)>
<!ELEMENT condition (#PCDATA)>
<!ELEMENT directive (#PCDATA)>
<!ELEMENT action (#PCDATA)>
```

The XML tags in the controller definition are explained below:

<xml encoding="US-ASCII"?>
Processing instruction that indicates that US-ASCII is used as the encoding for the XML content model for the controller.

<!ELEMENT statemachine (name, directive?, state+)>
The controller is essentially a state machine that has a unique name, an optional directive element, and one or more states (at least one state). The name of an Adaptive Document controller must match the name of the adaptive document (for example, SolutionAdoc) and the name of the Activity controller must match the name of the Activity (for example, AI).

<!ATTLIST statemachine id ID #REQUIRED>
An identifier, for example SolutionAdoc, that identifies the state machine.

<!ELEMENT state (name,type?, transition*)>
Each state in the controller has a name (unique in the scope of the controller), an optional type, such as Normal, and zero or more transitions that it originates. When an adaptive document controller is initialized, it is in an Open state. When it is in initialized, an Activity Controller is in the available state.

<!ELEMENT name (#PCDATA)>
The name element is used in defining the state and the state machine elements.

<!ELEMENT type (#PCDATA)>
The optional type element is used in defining the state element.

<!ELEMENT transition (target, event?, condition?, directive?, action*)>
The transition element defines the transition from a state to a target

state, given an event that is optional, subject to satisfying an optional guard condition, applying an optional directive, and triggering zero or more actions as part of the transition.

<!ELEMENT target (#PCDATA)>
The target state element that is used to define the transition element.

<!ELEMENT event (#PCDATA)>
The event element that defines the context for the transition. See the service request in "Chapter 3. Client programming" on page 33 to understand the importance of this element.

<!ELEMENT condition (#PCDATA)>
Condition, an optional element, is used in defining the transition element. It is evaluated prior to effecting a transition. More details on the usage of the condition element are explained in "Conditional logic in controllers" on page 17.

<!ELEMENT directive (#PCDATA)>
The Directive is an optional element that is used to express additional processing instructions for the controller. The Directive element is applied either at the controller level or in a transition. The Directive at the lowest level supersedes the directive at any higher level. Currently only the AUDIT directive is recognized by the controller. Examine the following controller definition that contains the AUDIT directive.

```
<?xmlversion="1.0"?>
<!DOCTYPE statemachine SYSTEM "controller.dtd">
<statemachine id="RegistrationAdoc">
 <name>RegistrationAdoc</name>
 <directive>AUDIT:NO</directive>
  <state>>
    <name>Open</name>
    <type>Normal</type>
    <transition>
       <target>Verifying</target>
       <event>Create</event>
       <condition></condition>
       <directive>AUDIT:YES</directive>
       <action>CreateNewRegistration</action>
    </transition>
 </state>
</statemachine>
```

In the above example, the first AUDIT directive applies to the overall controller definition while the second directive applies to the transition from Open state to the Verifying state. In this case the transition is logged because the transition level directive supersedes the one at the controller level. Further, the controller level AUDIT directive supersedes the AUDIT flag if any, that is set in the LDAP. The AUDIT flag in LDAP applies to the all of Process Broker Services.

<!ELEMENT action (#PCDATA)>

The Action element represents the commands that are invoked within a transition. See "Command definition" on page 19 for more details on the usage of this tag.

Adaptive document controller

The Adaptive Document Controller follows the controller definition. The Adaptive Document Controller by default starts in the Open state. Adaptive document controllers are associated with adaptive documents as described in "Creating and configuring controllers" on page 10.

A simple Adaptive Document Controller example is shown as a state chart and in XML form in Figure 5.

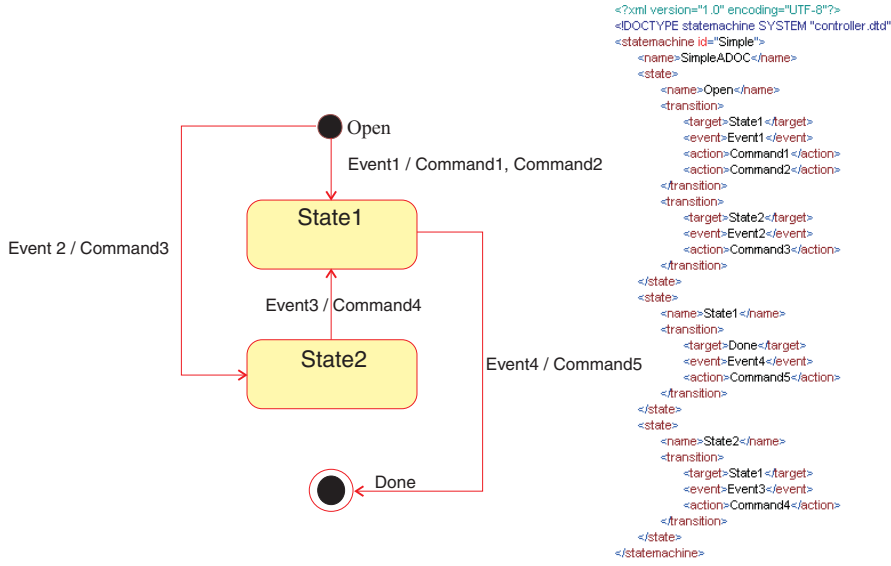

Figure 5. Simple Adaptive Document Controller example

Activity controllers

The Activity Controller, like the Adaptive Document Controller, follows the controller definition. However, the Activity Controller by default starts in the Available state. Further, the events in the Activity controller follow the naming convention ActivityName.Event, where the ActivityName must be replaced with the name of the Activity. Activity controllers are associated with Activities as described in "Creating and configuring controllers" on page 10.

If an Activity Controller is not defined for an Activity, then Process Broker Services associates the defaultActivityController to the Activity. The defaultActivityController definition is provided in the next example:

```
<?xml version="1.0" encoding="UTF-8">
<!DOCTYPE statemachine SYSTEM "controller.dtd">
<statemachine id="Default">
<name>defaultActivity</name>
<state>
  <name>Available</name>
    <transition>
        <target>Claimed</target>
        <event>DefaultActivity.Claim</event>
        <action>WWWFS_Claim_2</action>
    </transition>
 </state>
 <state>
   <name>Claimed</name>
   <transition>
     <target>Completed</target>
     <event>DefaultActivity.Complete</event>
     <action>WWFS_Complete_2</action>
   </transition>
 </state>
 </statemachine>
```

Conditional logic in controllers

Conditions are used in the definition of transitions in the Adaptive Document and Activity controllers. Conditions are expressed by specifying the name-value pairs that are delimited by a semi-colon; {<name>:<value>;<name>:<value>;...}. For example: **<condition>USER:John Doe;SHIPPER:Acme Inc</condition>**

In this example, the USER and SHIPPER element values are obtained from the context hashtable that is part of the Process Broker Services service request and compared to the values provided in the condition (See "Process Broker Services Web clients" on page 33 for details on the service request). If the conditions match, the transition is triggered.

Conditions are used to also express role-based access to specific transitions in the Adaptive Document and Activity controllers. The controller checks for the ROLES element and its value in either the input or the context hashtable in the service request. There are different ways of expressing Role-based conditional logic:

<condition></condition>
: This statement implies that the transition and the associated actions in the transition is role independent.

<condition>ROLES:Vendor</condition>
: Only the Vendor role has access to the transition and the privileges to invoke the actions associated with the transition.

<condition>ROLES:Vendor|Supplier</condition>
Either the Vendor role or the Supplier role has access to the transition and the privileges to invoke the actions associated with the transition. Here the character "|" stands for OR.

See "Process Broker Services Web clients" on page 33 for the specific Process Broker Services Interface methods that provides role-based queries, such as the list of permissible events for a given role.

User defined conditional logic

User defined conditional logic can be used in defining the transitions in either the Adaptive Document controllers or the Activity Controllers. This requires implementing a class that captures the conditional logic, say MyConditionClass, and referencing this in the condition expression with the keyword USERCOND such as:

<condition>USERCOND:MyConditionClass</condition>

The MyConditionClass must implement the evaluate method in the `com.ibm.epic.bfm.controller.UserCondition` interface. The signature of the evaluate method is as follows:

Boolean *evaluate* **(String request, Hashtable context, Hashtable input)**

The evaluate method returns a Boolean value, for example, true or false. The input parameters to the evaluate method include the request (for example, the event), the context, and the input that are passed by the service request to Process Broker Services. See "Process Broker Services Web clients" on page 33.

As an example, consider a scenario where the user defined condition is to check if the purchase order amount is greater than a certain amount. The MyConditionClass is implemented as follows.

```
Class MyConditionClass implements com.ibm.epic.bfm.controller.UserCondition {

  public MyConditonClass() {
       super();
  }
  Boolean evaluate(String request,Hashtable context, Hashtable input)
  {
    String poAdocId = (String) input.get ("ADOCREFERNCE");
    POAdoc poAdoc = POAdocHome.findByPrimaryKey(new AdocKey(poAdocId));
    String poBOId = poAdoc.getBOId();
    //The POService is a session bean – include the JNDI lookup and create logic here
    long poAmount = POService.getPOAmount (poBOId);
    if (poAmount > 100) return Boolean.TRUE;
    return Boolean.FALSE;
  }
}
```

The MyConditionClass requires a constructor with no arguments to be provided. This MyConditionClass is then used as part of the transition logic.

See "Chapter 4. Advanced Topics" on page 47 for additional information on use of the conditional logic in non-deterministic scenarios. For example, when the same event can cause different transitions to be invoked from a given state based on the conditional logic.

Command definition

The definition of a command (actions on a transition in either the Adaptive Document controller or the Activity Controller) is based on the command XML Data Type Definition (command.dtd):

```
<?xml encoding="US-ASCII"?>
<!ELEMENT commandlist (command+)>
<!ELEMENT command (input*, output*, methodName, receiverId, undoCmd*)>
<!ATTLIST command id ID #REQUIRED>
<!ELEMENT input (name, value?)>
<!ELEMENT output (name, value?)>
<!ELEMENT name (#PCDATA)>
<!ELEMENT value (#PCDATA)>
<!ELEMENT methodName (#PCDATA)>
<!ELEMENT receiverId (#PCDATA)>
```

<?xml encoding="US-ASCII"?>
Processing instruction that indicates that US-ASCII is used as the encoding for the content model for the command.

<!ELEMENT commandlist (command+)>
Process Broker Services has a list of commands that are defined in the command.xml file.

<!ELEMENT command (input*, output*, methodName, receiverId, undoCmd*)>
A command element is defined with zero or more input and output elements, a method name, the receiver identifier for the command, and zero or more undo commands.

<!ATTLIST command id ID #REQUIRED>
Each command is identified uniquely with the command id attribute.

<!ELEMENT input (name, value?)>
An input element for a command consists of the name, value pair where the value is optional.

<!ELEMENT output (name, value?)>
The output element, similar to the input, also consists of the name, value pair where the value is optional.

<!ELEMENT name (#PCDATA)>
The name element is used in defining the input and output elements.

<!ELEMENT value (#PCDATA)>
The value element, an optional element, is used in defining input and output for a command.

<!ELEMENT methodName (#PCDATA)>
The methodName element along with the input and output represents the signature of the command.

<!ELEMENT receiverId (#PCDATA)>
The receiverId identifies the implementation for a command. For example, the unique receiver that actually executes the command.

A command.xml file with two commands is shown in the next example. Both commands use the same receiver, but invoke different methods with appropriate input parameters.

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE commandlist SYSTEM "command.dtd">
<commandlist>
<command id="CreateNewRegistration">
<input>
<name>RegistrationType</name>
<value/>
</input>
<name>RegistrationInputList</name>
<value/>
</input>
<methodName>newRegistration</methodName>
<receiverId>RegistrationRMIBOReceiver</receiverId>
</command>
<command id="ApproveNotification">
<input>
<name>RegistrationId</name>
<value/>
</input>
<name>RegistrationType</name>
<value/>
</input>
<methodName>sendApproveNotifyMessage</methodName>
<receiverId>RegistrationRMIBOReceiver</receiverId>
</command><><>
```

Command groups

Commands can also be grouped into a command group. Command groups are used to batch commands that are sequentially executed. For any other complex sequencing of commands, it is advisable to compose them as a microflow using the microflow builder. The command group XML data type definition follows:

```
<?xml encoding="US-ASCII"?>
<!ELEMENT commandGrplist (commandGroup+)>
<!ELEMENT commandGroup (command+)>
<!ATTLIST commandGroup id ID #REQUIRED>
<!ELEMENT command (#PCDATA)>
```

<?xml encoding=″US-ASCII″?>
Processing instruction that indicates that US-ASCII is used as the encoding for the XML content model for the command group.

<!ELEMENT commandGrplist (commandGroup+)>
The command groups are defined in the commandGroup.xml file in Process Broker Services (This file is registered in LDAP, see Figure 4 on page 13.

<!ELEMENT commandGroup (command+)>
A command group contains one or more commands that are sequentially executed.

<!ATTLIST commandGroup id ID #REQUIRED>
Each command group is uniquely identified by its id attribute.

<!ELEMENT command (#PCDATA)>
The command element is used to define the command group.

A sample commandGroup.xml file is shown below:

```
<?xml version="1.0"?>
<!DOCTYPE commandGrplist SYSTEM "commandGroup.dtd">
<commandGrplist>
  <commandGroup id="InitializeSolutionAdoc">
    <command>getSolutionBOId</command>
    <command>setAdocBOId</command>
    <command>setSolutionBOFieldabc</command>
  </commandGroup>
 </commandGrplist>
```

In this example, the InitializeSolutionAdoc command group represents a sequence of three commands. This command group can be used as an action in the controller definition just as any individual command.

Process Broker Services system commands

Process Broker Services has commands that are pre-defined in the command.xml file. These are referred to as the system commands. Process Broker Services System commands are categorized as the scheduler system commands for invoking the Process Broker Services scheduler, and are categorized as the workflow system commands for invoking WebSphere Workflow Services.

Scheduler system commands: Scheduler system commands are pre-defined commands that invoke the services of the Process Broker Services Scheduler. There are nine scheduler system commands. Each command consists of a

method name, a receiver id, and a number of input and output parameters. Figure 6 provides a diagram of one of the scheduler system commands.

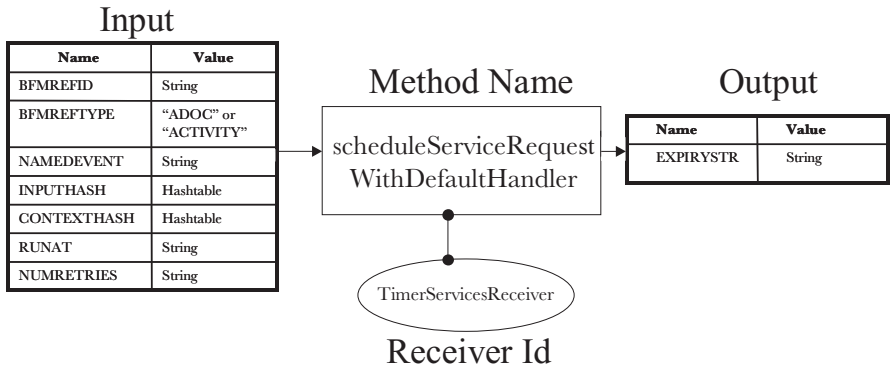

Figure 6. Scheduler System Command Example

The XML for the scheduler system command shown in Figure 6, is based on the command.dtd:

```
<command id="PBS_ScheduleServiceRequestWithDefaultHandler">
     <!-- the adoc or activity id, String -->
     <input><name>BFMREFID</name><value></value></input>
     <!-- ADOC or ACTIVITY, String -->
     <input><name>BFMREFTYPE<value>ADOC</value></input>
     <!-- The Event to be raised, String -->
     <input><name>NAMEDEVENT</name><value></value></name>
     <!-- The input hashtable, Hashtable -->
     <input><name>INPUTHASH</name><value></value></input>
     <!-- The context hashtable, Hashtable -->
     <input><name>CONTEXTHASH</name><value></value></input>
     <!-- Date/time, String -->
     <input><name>RUNAT</name></value></input>
     <!-- Number of Retries, String -->
     <input><name>NUMRETRIES</name><value></value></input>
     <!--The expiry date/time, String-->
     <output><name>EXPIRYSTR</name><value></value></output>
     <methodName>schduleServiceRequestWithDefaultHandler</methodName>
     <receiverId>TimerServicesReceiver</receiverId>
     </command>
```

This scheduler system command is used to schedule service requests with a specific handler. See "Process Broker Services scheduler" on page 50 for details on writing a customized handler.

```
<command id="PBS_ScheduleServiceRequestWithGivenHandler">
    <input><name>BFMREFID</name><value></value></input>
    <input><name>BFMREFTYPE</name><value>ADOC</value></input>
    <input><name>NAMEDEVENT</name><value>Timeout</value></input>
```

```
    <input><name>INPUTHASH</name><value></value></input>
    <input><name>CONTEXTHASH</name><value></value></input>
    <!-- The fully qualified handler class, String -->
    <input><name>HANDLERCLASS</name><value></value></input>
    <input><name>RUNAT</name><value></value></input>
    <input><name>NUMRETRIES</name><value></value></input>
    <output><name>EXPIRYSTR</name><value></value></output>
    <methodName>scheduleServiceRequestWithGivenHandler</methodName>
    <receiverId>TimerServicesReceiver</receiverId>
</command>
```

This system command is used to schedule a service request with a specific handler and on behalf of a given user. Such a request is meaningful especially when there are role-based conditions on a transition in the controller.

```
<command id="PBS_ScheduleServiceRequestWithGivenHandlerBySpecifiedUser">
    <input><name>BFMREFID</name><value></value></input>
    <input><name>BFMREFTYPE</name><value>ADOC</value></input>
    <input><name>NAMEDEVENT</name><value>Timeout</value></input>
    <input><name>INPUTHASH</name><value></value></input>
    <input><name>CONTEXTHASH</name><value></value></input>
    <input><name>HANDLERCLASS</name><value></value></input>
    <input><name>RUNAT</name><value></value></input>
    <input><name>NUMRETRIES</name><value></value></input>
    <!-- User, String -->
    <input><name>USER</name><value></value></input>
    <output><name>EXPIRYSTR</name><value></value></output>
    <methodName>scheduleServiceRequestWithGivenHandlerBySpecifiedUser</methodName>
    <receiverId>TimerServicesReceiver</receiverId>
</command>
```

This system command is used to schedule a service request given the PBSEventInput object and using the default action listener or handler. (See “Chapter 3. Client programming” on page 33 for making service requests by passing a Process Broker Services event object.)

```
<command id="PBS_ScheduleServiceRequestWithDefaultHandlerPBSInput">
   <!-- The PBSEventInput, PBSEventInput -->
   <input><name>PBSEVENTINPUT</name><value/></input>
   <input><name>RUNAT</name><value/></input>
   <input><name>NUMRETRIES</name><value/></input>
   <output><name>EXPIRYSTR</name><value/></output>
   <methodName>scheduleServiceRequestWithDefaultHandler</methodName>
   <receiverID>TimerServicesReceiver</receiverId>
</command>
```

This system command is used to schedule a service request given the PBSEventInput object and using a given action listener or handler. (See “Chapter 3. Client programming” on page 33 for making a service request by passing a Process Broker Services event object.)

```
<command id="PBS_ScheduleServiceRequestWithGivenHandlerPBSInput">
      <input><name>PBSEVENTINPUT</name><value/></input>
      <input><name>HANDLERCLASS</name><value/></input>
```

```
      <input><name>RUNAT</name><value/></input>
      <input><name>NUMRETRIES</name><value/></input>
      <output><name>EXPIRYSTR</name><value/></output>
      <methodName>scheduleServiceRequestWithGivenHandler</methodName>
      <receiverId>TimerServicesReceiver</receiverId>
</command>
```

This system command is used to schedule a service request to initialize an adaptive document that is created with a system generated adaptive document Id.

```
<command id="PBS_ScheduleAdocInitializationWithGeneratedId">
      <input><name>AC_ADOCTYPE</name><value></value></input>
      <input><name>AC_ADOCOWNER</name><value></value></input>
      <input><name>AC_NAMEDEVENT</name><value>Initialize</value></input>
      <input><name>AC_INPUTHASH</name><value></value></input>
      <input><name>AC_CONTEXTHASH</name><value></value></input>
      <input><name>AC_RUNAT</name><value></value></input>
      <input><name>AC_NUMRETRIES</name><value></value></input>
      <output><name>AC_EXPIRYSTR</name><value></value></output>
      <methodName>scheduleAdocInitializationWithGeneratedId</methodName>
      <receiverId>TimerServicesReceiver</receiverId>
</command>
```

This system command is used to schedule a service request to initialize an adaptive document that is created with a given adaptive document id.

```
<command id="PBS_ScheduleAdocInitializationWithGivenId">
      <input><name>AC_ADOCTYPE</name><value></value></input>
      <input><name>AC_ADOCOWNER</name><value></value></input>
      <input><name>AC_NAMEDEVENT</name><value>Initialize</value></input>
      <input><name>AC_INPUTHASH</name><value></value></input>
      <input><name>AC_CONTEXTHASH</name><value></value></input>
      <input><name>AC_RUNAT</name><value></value></input>
      <input><name>AC_NUMRETRIES</name><value></value></input>
      <output><name>AC_EXPIRYSTR</name><value></value></output>
      <methodName>scheduleAdocInitializationWithGivenId</methodName>
      <receiverId>TimerServicesReceiver</receiverId>
</command>
```

This system command is used to schedule a service request for archiving an adaptive document.

```
<command id="PBS_ScheduleAdocArchival">
      <input><name>AR_ADOCID</name><value></value></input>
      <input><name>AR_RUNAT</name><value></value></input>
      <input><name>AR_NUMRETRIES</name><value></value></input>
      <output><name>AR_EXPIRYSTR</name><value></value></output>
      <methodName>scheduleAdocArchival</methodName>
      <receiverId>TimerServicesReceiver</receiverId>
</command>
```

This system command is used to schedule a service request to remove an adaptive document.

```
<command id="PBS_ScheduleAdocRemoval">
      <input><name>RE_ADOCID</name><value></value></input>
      <input><name>RE_RUNAT</name><value></value></input>
      <input><name>RE_NUMRETRIES</name><value></value></input>
      <output><name>RE_EXPIRYSTR</name><value></value></output>
      <methodName>scheduleAdocRemoval</methodName>
      <receiverId>TimerServicesReceiver</receiverId>
</command>
```

Workflow system commands: The workflow system commands are used to communicate to the workflow engine using the WebSphere Workflow Services layer. These commands take a number of input parameters such as the PROCDEF, INSTANCENAME, ACTIVITYVARS, PROCVARS, and USER. These input parameters are automatically initialized by Process Broker Services and do not have to be provided when using the workflow system commands.

A diagram of an example workflow system command is shown in Figure 7. There are nine workflow system commands, all based on the command.dtd.

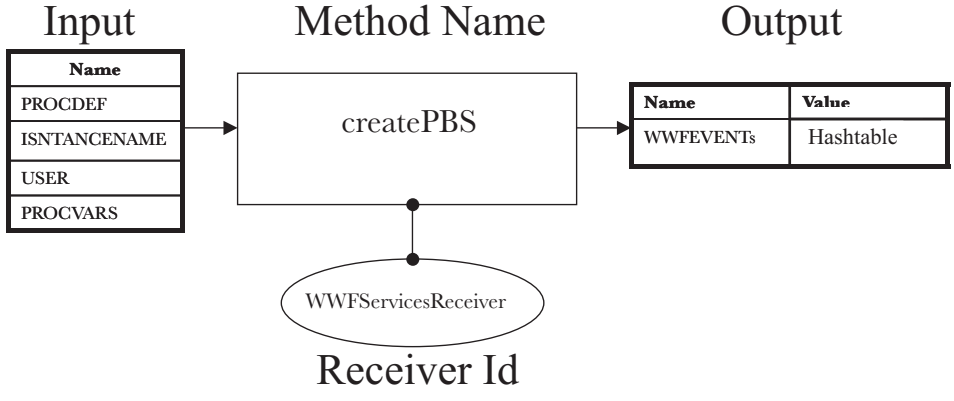

Figure 7. Workflow System Command Example

The command definition in XML for the sample Workflow system command shown in Figure 7 is as follows. It is used to create a process instance in the underlying workflow engine.

```
<command id="WWFS_Create">
      <input><name>PROCDEF</name><value></value></input>
      <input><name>INSTANCENAME</name><value></value></input>
      <input><name>USER</name><value></value></input>
      <input><name>PROCVARS</name><value></value></input>
```

```
      <output><name>WFEVENTs</name><value></value></output>
      <methodName>createPBS</methodName>
      <receiverId>WWFServicesReceiver</receiverId>
</command>
```

This workflow system command is used to claim an activity that is available for a given process and a user.

```
<command id="WWFS_Claim_1">
      <input><name>PROCDEF</name><value></value></input>
      <input><name>INSTANCENAME</name><value></value></input>
      <input><name>USER</name><value></value></input>
      <input><name>ACTIVITYVARS</name><value></value></input>
      <output><name>WFEVENTs</name><value></value></output>
      <methodName>claimPBS</methodName>
      <receiverId>WWFServicesReceiver</receiverId>
</command>
```

The following system command is used to claim a specific activity in a process that is available for a given user.

```
<command id="WWFS_Claim_2">
      <input><name>PROCDEF</name><value></value></input>
      <input><name>INSTANCENAME</name><value></value></input>
      <input><name>ACTIVITYNAME</name><value></value></input>
      <input><name>USER</name><value></value></input>
      <input><name>ACTIVITYVARS</name><value></value></input>
      <output><name>WFEVENTs</name><value></value></output>
      <methodName>claimPBS</methodName>
      <receiverId>WWFServicesReceiver</receiverId>
</command>
```

The command to complete an activity claimed using the system command is in the following example:

```
<command id="WWFS_Complete_1">
      <input><name>PROCDEF</name><value></value></input>
      <input><name>INSTANCENAME</name><value></value></input>
      <input><name>USER</name><value></value></input>
      <input><name>ACTIVITYVARS</name><value></value></input>
      <output><name>WFEVENTs</name><value></value></output>
      <methodName>completePBS</methodName>
      <receiverId>WWFServicesReceiver</receiverId>
</command>
```

The command to complete the activity claimed using the system command is in the following example:

```
<command id="WWFS_Complete_2">
      <input><name>PROCDEF</name><value></value></input>
      <input><name>INSTANCENAME</name><value></value></input>
      <input><name>ACTIVITYNAME</name><value></value></input>
      <input><name>USER</name><value></value></input>
      <input><name>ACTIVITYVARS</name><value></value></input>
```

```
      <output><name>WFEVENTs</name><value></value></output>
      <methodName>completePBS</methodName>
      <receiverId>WWFServicesReceiver</receiverId>
</command>
```

The command to unclaim an activity that is claimed using the system command `command id="WWFS_Complete_1"`

```
<command id="WWFS_Unclaim_1">
      <input><name>PROCDEF</name><value></value></input>
      <input><name>INSTANCENAME</name><value></value></input>
      <input><name>USER</name><value></value></input>
      <input><name>ACTIVITYVARS</name><value></value></input>
      <output><name>WFEVENTs</name><value></value></output>
      <methodName>unclaimPBS</methodName>
      <receiverId>WWFServicesReceiver</receiverId>
</command>
```

The command to unclaim an activity that is claimed using the system command `command id="WWFS_Complete_2"`

```
<command id="WWFS_Unclaim_2">
      <input><name>PROCDEF</name><value></value></input>
      <input><name>INSTANCENAME</name><value></value></input>
      <input><name>ACTIVITYNAME</name><value></value></input>
      <input><name>USER</name><value></value></input>
      <input><name>ACTIVITYVARS</name><value></value></input>
      <output><name>WFEVENTs</name><value></value></output>
      <methodName>unclaimPBS</methodName>
      <receiverId>WWFServicesReceiver</receiverId>
</command>
```

This is the command to force the completion of a specific activity for a given process.

```
<command id="WWFS_ForceFinish">
      <input><name>PROCDEF</name><value></value></input>
      <input><name>INSTANCENAME</name><value></value></input>
      <input><name>ACTIVITYNAME</name><value></value></input>
      <input><name>USER</name><value></value></input>
      <input><name>ACTIVITYVARS</name><value></value></input>
      <output><name>WFEVENTs</name><value></value></output>
      <methodName>ForceFinishPBS</methodName>
      <receiverId>WWFServicesReceiver</receiverId>
</command>
```

This is the command to terminate a given process.

```
<command id="WWFS_Terminate">
      <input><name>PROCDEF</name><value></value></input>
      <input><name>INSTANCENAME</name><value></value></input>
      <input><name>USER</name><value></value></input>
      <input><name>ACTIVITYVARS</name><value></value></input>
```

```
      <output><name>WFEVENTs</name><value></value></output>
      <methodName>TerminatePBS</methodName>
      <receiverId>WWFServicesReceiver</receiverId>
</command>
```

Receiver definitions

Receivers are implementations of the commands that are used in the Adaptive Document and Activity Controllers. Each command is associated with a unique receiver that is identified by its receiver id. See "Command definition" on page 19. A receiver, however, can provide implementation for multiple commands. Receivers in Process Broker Services are defined in the receiver.xml file based on the receiver XML data type definition (receiver.dtd).

The receiver.dtd is shown next. The associated transport definition follows the receiver dtd.

```
<?xml encoding="US-ASCII"?>
<!ENTITY % PROTOCOL SYSTEM "transport.dtd">
%PROTOCOL;
<!ELEMENT receiverlist (receiver)+>
<!ELEMENT receiver (mode)>
<!ATTLIST receiver id ID #REQUIRED>
<!ELEMENT mode (protocol)>
```

<?xml encoding="US-ASCII"?>
Processing instruction that indicates that US-ASCII is used as the encoding for the XML definition for the receiver.

<!ENTITY % PROTOCOL SYSTEM "transport.dtd">
The data structure for the protocol associated with receivers is defined in the transport.dtd file.

%PROTOCOL;
The transport.dtd definition is included to complete the receiver definition.

<!ELEMENT receiverlist (receiver)+>
The list of receivers, one or more, in Process Broker Services is defined in the receiver.xml file.

<!ELEMENT receiver (mode)>
Each receiver element consists of the mode element in its definition.

<!ATTLIST receiver id ID #REQUIRED>
The receiver id attribute uniquely identifies a receiver.

<!ELEMENT mode (protocol)>
The mode element consists of the protocol, whose structure is defined in the transport.dtd.

```
<?xml encoding="US-ASCII"?>
<!ELEMENT protocol (iiop | rmi | localrmi | native)>
<!ELEMENT iiop (objectIORfile, JNDIname, Home)>
```

```
<!ELEMENT rmi (providerHost?, providerPort?, initialContext, JNDIname, Home,
                  PKclassName?, PKParamName*)>
<!ELEMENT native (JNDIname, PKParamName*)>
<!ELEMENT localrmi (JNDIname, PKclassName?, PKParamName*)>
<!ELEMENT providerHost (#PCDATA)>
<!ELEMENT providerPort (#PCDATA)>
<!ELEMENT objectIORfile (#PCDATA)>
<!ELEMENT initialContext (#PCDATA)>
<!ELEMENT JNDIname (#PCDATA)>
<!ELEMENT Home (#PCDATA)>
<!ELEMENT PKclassName (#PCDATA)>
<!ELEMENT PKParamName (name, value?)>
<!ELEMENT name (#PCDATA)>
<!ELEMENT value (#PCDATA)>
```

<?xml encoding="US-ASCII"?>
Processing instruction that indicates that US-ASCII is used as the encoding for the XML definition for the receiver protocol.

<!ELEMENT protocol (iiop | rmi | localrmi | native)>
Currently the protocols for the receivers include iiop, rmi, localrmi, and native.

- The Internet Inter-ORB Protocol (iiop) is a TCP/IP specific CORBA standard transport protocol and is used to communicate to remote CORBA objects.
- The Remote Method Invocation (rmi) protocol is a Java-only version of the CORBA standard and is used to communicate to remote Java objects.
- The localrmi is also a Remote Method Invocation protocol. In localrmi and rmi, Process Broker Services caches the home reference of the object and does not look up the receiver upon every invocation as it does for receivers. The localrmi is typically the protocol used for the system receivers in Process Broker Services that are operating in the same Java Virtual Machine as Process Broker Services.
- The native protocol is used to communicate to simple Java objects in the same Java Virtual Machine as Process Broker Services.

Note: Note that Process Broker Services can use microflows as either rmi or native receivers to compose XML messages and send these to various end points using the MQseries Adapter Kernel infrastructure.

<!ELEMENT iiop (objectIORfile, JNDIname, Home)>
The objectIORfile, JNDIname, and the Home parameters define the iiop protocol.

<!ELEMENT rmi (providerHost?, providerPort?, initialContext, JNDIname, Home, PKclassName?, PKParamName*)>
The rmi protocol is defined by a collection of parameters of which the

providerHost, *providerPort*, and the *PKClassName* are optional elements. The *initialContext*, *JNDIname*, and *Home* are mandatory elements in the rmi protocol. There are also zero or more *PKParamName* elements in the rmi protocol. Note that the *PKClassName* and the *PKParamName* are both essential to create the *Home* object if the rmi receiver is a container-managed persistence entity bean. In the case that the rmi receiver is a session bean only, the *PKParamName* is needed for defining the rmi protocol. Process Broker Services in WebSphere Business IntegratorVersion 2.1 only supports rmi receivers for Enterprise Beans as target objects.

An example of an rmi protocol receiver is shown below.

```
<receiver id="WWFServicesReceiver">
   <mode>
    <protocol>
      <rmi>
       <providerHost>9.83.96.192</providerHost>
       >providerPort<900></providerPort>
       <initialContext>com.ibm.ejs.ns.jndi.CNInitialContextFactory</initialContext>
             <JNDIname>com.ibm.b2bi.bfm.ejb.WWFServices</JNDIname>
             <Home>com.ibm.b2bi.bfm.ejb.WWFServicesHome</Home>
       </rmi>
      </protocol>
     </mode>
</receiver>
```

<!ELEMENT native (JNDIname, PKParamName*)>

The native protocol uses the *JNDIname* and the *PKParamName* (zero or more) as parameters. The *PKParamName* contains the input that is required to create the target object. An example of a native protocol receiver is shown below.

```
<receiver id="LogAdapterReceiver">
      <mode>
         <protocol>
            <native>
              <JNDIname>com.ibm.epic.LogTrace.EpicLog</JNDIname>
              <PKParamName>
                  <name>appName</name>
                  <value/>
              </PKParamName>
              <PKParamName>
                  <name>compName</name>
                  <value>BFM</value>
              </PKParamName>
           </native>
        </protocol>
     </mode>
</receiver>
```

<!ELEMENT localrmi (JNDIname, PKclassName?, PKParamName*)>

The localrmi protocol uses the *JNDIname*, an optional *PKclassName* and zero or more *PKParamName*s as parameters. For localrmi-based receivers

the home is obtained from the Process Broker Services cache. As in the case of the rmi receivers, the localrmi receivers in Process Broker Services for WebSphere Business IntegratorVersion 2.1 only support Enterprise Beans as target objects. An example localrmi protocol receiver is shown below.

```
<receiver id="TimerServicesReceiver">
      <mode>
         <protocol>
            <localrmi>
              <JNDIname>com.ibm.epic.bfm.TimerService</JNDIname>
            </localrmi>
        </protocol>
     </mode>
</receiver>
```

<!ELEMENT providerHost (#PCDATA)>
The *providerHost* optional element is used in the definition of the rmi protocol based receiver.

<!ELEMENT providerPort (#PCDATA)>
The *providerPort* is also an optional element that is used in the definition of the rmi protocol based receiver.

<!ELEMENT objectIORfile (#PCDATA)>
The *objectIORfile* is used as a parameter in defining the iiop protocol based receiver.

<!ELEMENT initialContext (#PCDATA)>
The *initialContext* is a parameter used in defining the rmi protocol based receiver.

<!ELEMENT JNDIname (#PCDATA)>
The *JNDIname* element is used as a parameter in rmi, localrmi, and native protocol receiver definitions.

<!ELEMENT Home (#PCDATA)>
The *Home* element is used in defining the rmi protocol.

<!ELEMENT PKclassName (#PCDATA)>
The *PKclassName* is an optional element that is used in defining the rmi and the localrmi protocol.

<!ELEMENT PKParamName (name, value?)>
The *PKParamName* element consists of a name and an optional value element. The PKParamName is used in defining rmi, localrmi, and native protocols – it represents parameters that are typically used in the Primary Key class constructor.

<!ELEMENT name (#PCDATA)>
The name element is used in defining the PKParamName.

<!ELEMENT value (#PCDATA)>
The value element is an optional element also used in the PKParamName.

System receivers

Some receivers are pre-defined in the receiver.xml file and are provided with Process Broker Services. These receivers are referred to as the System Receivers and they include:

WWFServicesReceiver
Use the WWFServicesReceiver to communicate with any JointFlow based workflow engine. This receiver executes workflow actions on WebSphere Workflow Services. It is defined as an rmi based protocol receiver. See the rmi protocol description and the sample rmi protocol receiver on page 29.

TimerServicesReceiver
Use the TimerServicesReceiver to execute actions on the Process Broker Services Scheduler. It is defined as a localrmi based protocol receiver. See the sample localrmi protocol receiver on page 31.

LogAdapterReceiver
Use the LogAdapterReceiver to execute actions on the Business Flow Manager solution management services to generate audit and exception logs from Process Broker Services. It is defined as a native protocol based receiver. See the sample LogAdapterReceiver on page 30.

Chapter 3. Client programming

The clients can communicate with Process Broker Services either using remote method invocation (Web Clients), or by sending messages (Messaging Clients). In either case, the Process Broker Services Interface provides the set of services that the clients use to interact with adaptive documents (See "Appendix B. Process Broker Services Application Program Interfaces" on page 69 for details on the Process Broker Services Application Programming Interface).

Process Broker Services Web clients

Typically users with web browsers interact with Process Broker Services using the Interaction Manager that maintains the user session and renders the content delivered by Process Broker Services (see the *WebSphere Business Integrator Run Time* book for more details on Interaction Manager). The Process Broker Services Web Client, therefore, refers to the Interaction Manager and any other web application that interacts with Process Broker Services. The Process Broker Services Web Client Interaction Pattern is shown in Figure 8 on page 34.

1. The Process Broker Services Web Clients access the Process Broker Services interface through the Business Flow Manager Access Bean, which is a client to the BFMAdmin Enterprise Bean that implements the Process Broker Services Interface. (WebSphere Business Integrator Version 2.1provides the bfm.client.jar and the bfm.samples.client.jar files as part of the Process Broker Services that can be used by Web Clients).
2. The Web Client interacts with Process Broker Services typically by making a serviceRequest (see "Composing service requests" on page 34 for details).
3. Process Broker Services then brokers this service request and raises the event on the appropriate adaptive document instance.
4. This event is consumed within one or more controllers associated with the adaptive document.
5. The event triggers transitions within the controllers causing them to launch one or more actions.
6. Receivers that represent the various endpoints execute these actions (business objects, applications, databases, and other enterprise information systems are examples of endpoints).
7. The actual communication with an endpoint uses a variety of communication protocols, such as, rmi, iiop, or MQ.
8. The output from the serviceRequest, following the adaptive document transitions, is then returned to the Web Client.

This pattern is also used by Web Clients to invoke other methods on the Process Broker Services interface such as adaptive document queries.

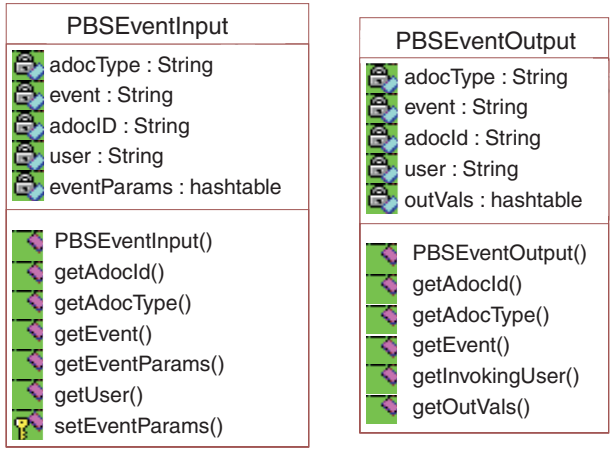

Figure 8. Web Client Interaction Pattern

Composing service requests

The most commonly used serviceRequest Application Programming Interface for Process Broker Services is:

```
public java.util.Hashtable serviceRequest (java.lang.String request,
                                           java.util.Hashtable context,
                                           java.util.Hashtable input,
                                           java.lang.String adocId,
                                           java.lang.String user)
                                           throws java.rmi.RemoteException
```

- The request refers to business events and is of type string. It corresponds to the events that are specified in the controller definitions using the `<event></event>` elements. Events on the Activity Controllers follow the convention ActivityName.eventName, where the ActivityName must be replaced by the actual name of the activity, and the eventName corresponds to the event name. The same event can be used in multiple controllers (see "Process brokering patterns" on page 52 for usage of this scenario). It is possible to query an adaptive document to get the events. The getAllAdocEvents Application Programming Interface call on Process Broker Services returns a list of adaptive documents and the events for each adaptive document (these are events on the Adaptive Document Controller). The getAllPossibleBusinessEvents API call on Process Broker Services returns all the events associated with an adaptive document instance given its current business state. For example, all the events that are

available for the set of Activity and Adaptive Document controllers that are associated with an adaptive document. (See "Querying adaptive documents" on page 39).

- The context refers to the context in which the service request is made and is of type Hashtable. The context parameters are name-value pairs in the Hashtable. The context parameters are used in the condition tags defined in the controllers (see "Conditional logic in controllers" on page 17 for details on condition logic). If no conditions are defined in a transition associated with an event, Process Broker Services ignores the context provided in the service request for that event.
- The input refers to the collection of input parameters that are required to execute the actions triggered by the service request and is of type Hashtable. The input parameters are ascertained by determining the list of actions and then inspecting the command and receiver definitions contained in the command.xml and receiver.xml files respectively.
- The adocId is the unique identifier of an adaptive document instance (it is equivalent to a business transaction identifier). The adocId can be obtained in several different ways prior to making the service request, and the particular approach that is used depends on the client scenario. For example, the createAdocIdReturn Application Program Interface creates a new adaptive document of a certain type and returns the adocId. The getAdocs Application Program Interface is used to obtain a vector of AdocDetails for a list of adaptive documents that match a certain criterion. The AdocDetails is a data structure that contains the base attributes of an adaptive document, such as the adocId, the adocName, the current adocOwner, and the current adocState.
- The user refers to the user who is making the service request. If the user is not known, then it is acceptable to pass a null value. The user information is used in the condition evaluation of a transition in the controller (see "Conditional logic in controllers" on page 17).
- The output from the service request is of type Hashtable and consists of the aggregated output from all the actions that were triggered by the service request on the appropriate controllers.

Service request with event object

Another form of serviceRequest is to compose an event object referred to as the PBSEventInput object and make the service request. The return value is the PBSEventOutput object.

```
public PBSEventOutput serviceRequest (PBSEventInput event)
                            throws java.rmi.RemoteException.
```

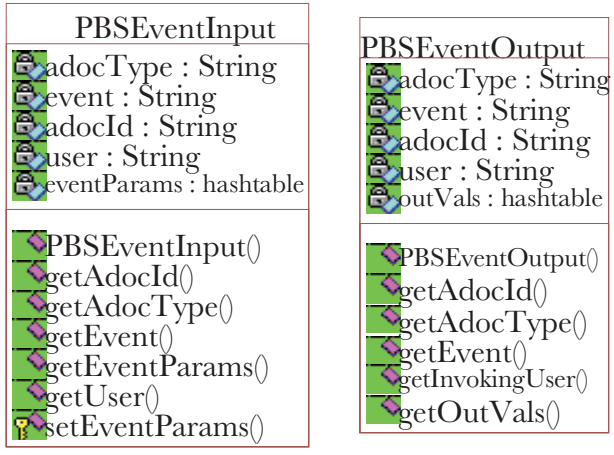

Figure 9. Process Broker Services Event Input and Output Objects for Service Requests

The PBSEventInput and the PBSEventOutput classes are shown in Figure 9. The PBSEventInput class contains the basic information necessary for making the service request. The eventParams Hashtable contains the event data, for example, the input and the context data (name-value parameter information) that is needed to make the service request. Note that the setEventParams is a protected method in the PBSEventInput class that is used to set the event data. The output data following a service request is contained in the outVals Hashtable in the PBSEventOutput class. The getOutVals method can be used to get the output data from the PBSEventOutput.

An example of how to use the PBSEventInput in a service request is provided in the test client in the com.ibm.epic.bfm.utils package that ships with WebSphere Business Integrator Version 2.1. This test client uses the TestAdoc in the com.ibm.epic.bfm.samples package.

- The example begins by creating a TestAdocIntializeEvent object that extends the PBSEventInput.

```
public class TestAdocInitializeEvent
                extends com.ibm.epic.bfm.ejb.base.PBSEventInput {

public TestAdocInitializeEvent(String adocId, String user) {
        //this essentially invokes the PBSEventInput constructor
        super("Test", "Initialize", adocId, user);}
}
```

- This event object is then used to make a service request. Notice that when the service request is invoked with a null in the adocId field, this implies that a new adaptive document is created.

```
TestAdocInitializeEvent ie = new TestAdocInitializeEvent(null, user);
PBSEventOutput o = getBFMSession().serviceRequest(ie);
```

- The service request with the Initialize event causes the following transition in the TestAdoc controller:

```
<state>
      <name>Open</name>
      <type>Normal</type>
      <transition>
          <target>Open</target>
          <event>Initialize</event>
          <condition/>
      </transition>
 </state>
```

- The TestAdoc is created and left in the Open state. The Intialize event triggers the command-group referred to as the InitializeTestAdoc. The individual commands in the command group are implemented by two receivers: the TestAdocReceiver and the TestBOReceiver.

```
<receiver id="TestAdocReceiver">
<mode>
<protocol>
<localrmi>
<JNDIname>com.ibm.epic.bfm.samples.TestAdoc</JNDIname>
<PKclassName>com.ibm.epic.bfm.samples.TestAdocKey</PKclassName>
<PKParamName>
<name>adocReference</name>
<value/>
</PKParamName>
</localrmi>
</protocol>
</receiver>
<receiver id="TestBOReceiver">
<mode>
<protocol>
<localrmi>
<JNDIname>com.ibm.epic.bfm.samples.TestBO</JNDIname>
<PKclassName>com.ibm.epic.bfm.samples.TestBOKey</PKclassName>
<PKParamName>
<name>testboid</name>
<value/>
</PKParamName>
</localrmi>
</protocol>
</mode>
</receiver>
```

- The adocId obtained from the PBSEventOutput is then used to make a subsequent service request with a new event object, TestAdocTimerTest_11Event.

```
public class TestAdocTimerTest_11Event
                extends com.ibm.epic.bfm.ejb.base.PBSEventInput {
public TestAdocTimerTest_11Event(String adocId, String user) {
  super("Test", "TimerTest_11", adocId, user);
  }
  public void setNumRetries(String numRetries) {
    super.setEventParam("NUMRETRIES", numRetries);
```

```
  }
  public void setPBSEventInputObject (com.ibm.epic.bfm.ejb.base.PBSEventInput p) {
    super.setEventParam("PBSEVENTINPUT", p);
  }
  public void setRunAt(String runat) {
    super.setEventParam("RUNAT", runat);
  }
}
```

- The service request with the TestAdocTimerTest_11Event object causes the TestAdoc to schedule a service request on the Process Broker Services Scheduler service. The TestAdocTimerTest_11Event object encapsulates the TimerTest_11 event that causes the TestAdoc to trigger the PBS_ScheduleServiceRequestWithDefaultHandler system command. The effect of this system command is to schedule a service request with a TimeOut event to be raised on the TestAdoc after 10 000 milliseconds and Process Broker Services can make four retries in case the event is not raised in the first attempt.

  ```
  TestAdocTimerTest_11Event event = new TestAdocTimerTest_11Event(o.getAdocId(), user);
  event.setNumRetries("4");
  event.setRunAt("10000");
  ```

- Consequently the event data in the TestAdocTimerTest_11Event object must contain the information necessary for the scheduling service system command. This information is contained in the TestAdocTimeoutEvent object that is then inserted as an event parameter in the TestAdocTimerTest_11Event object. The output Hashtable provides the output from the service request, in this case the EXPIRYSTR parameter (see "Process Broker Services system commands" on page 21).

  ```
  TestAdocTimeoutEvent toe = new TestAdocTimeoutEvent(o.getAdocId(), user);
  event.setPBSEventInputObject(toe.convertToPBSEventInput());
  o = getBFMSession().serviceRequest(event);
  Hashtable output = o.getOutVals();
  ```

Service request to create and initialize an adaptive document

The following Process Broker Services Application Program Interface creates an adaptive document and then raises an event through a subsequent service request to initialize the adaptive document.

```
public java.util.Hashtable initializeAdocDoServiceRequest
                                          (java.lang.String request,
                                           java.util.Hashtable context,
                                           java.util.Hashtable input,
                                           java.lang.String adocType,
                                           java.lang.String adocId,
                                           java.lang.String user)
                                           throws java.rmi.RemoteException
```

The parameters are the same as the basic service request. The request string represents the event that will be raised on the adaptive document after its creation. The adocId can be either null in which case Process Broker Services

generates a default Id, or the adocID can be explicitly provided for Process Broker Services to use in creating the adaptive document.

This Application Program Interface is particularly useful in initializing the adaptive document after its creation. The InitializeTestAdoc command group, introduced in "Service request with event object" on page 35, that is invoked upon raising the Initialize event on the TestAdoc, is an example where the initialization actions include: creating business objects, associating the references in the adaptive document, and then setting some business object attributes.

```
<commandGrplist>
   <commandGroup id="InitializeTestAdoc">
     <command>getTestBOId</command>
     <command>setAdocBOId</command>
     <command>setTestBOField1</command>
   </commandGroup>
</commandGrplist>

<command id="getTestBOId">
  <output><name>THETESTBOID</name><value></value></output>
  <methodName>getTestBOId</methodName>
  <receiverId>TestBOReceiver</receiverId>
</command>

<command id="setAdocBOId">
   <input><name>THETESTBOID</name><value></value></input>
   <methodName>setBoId</methodName>
   <receiverId>TestAdocReceiver</receiverId>
</command>

<command id="setTestBOField1">
   <input><name>THETESTBOID</name><value></value></input>
   <methodName>setBoField1</methodName>
   <receiverId>TestBOReceiver</receiverId>
</command>
```

Querying adaptive documents

The Web Clients can also use various Process Broker Services Application Program Interfaces to query adaptive documents. These Application Program Interface calls are categorized based on the scope of the query, either on a single adaptive document, or on a list of adaptive documents. Further, the queries are also classified based on the nature of the query. For example, to query the adaptive document details or the adaptive document services that are available based on its current state. A complete set of adaptive document queries is shown in Figure 10 on page 40.

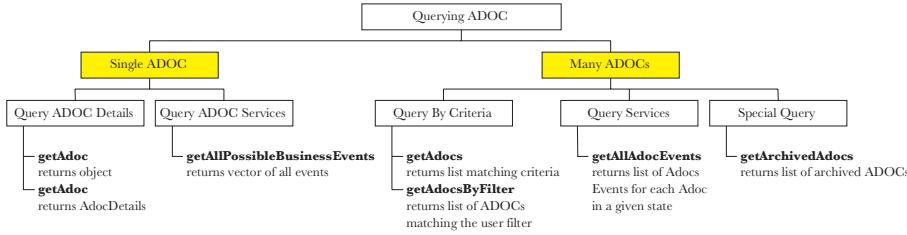

Figure 10. Adaptive document query Process Broker Services application program interface

Queries on a single adaptive document

- The getAdoc Application Program Interface returns either the adaptive document object or the AdocDetails containing the adaptive document attributes for a specific adaptive document instance.
- The getAllPossibleBusinessEvents Application Program Interface returns a vector of all business events (for example, services provided by an adaptive document) that are available for a given business state of the adaptive document, based on the state of the adaptive document and the Activity controllers associated with the adaptive document.

Queries yielding multiple adaptive documents

- The getAdocs Application Program Interface returns a vector of AdocDetails objects for any combination of type, user, and state information for an adaptive document.
- The Application Program Interface returns a vector of AdocDetails objects for a given user filter. The Solution adaptive documents have to implement the method "boolean filter (Hashtable filterParams)" for this Application Program Interface to be invoked. getAdocsByFilter is invoked in one of two ways:
  - with just the adaptive document type and the user-defined filter.
  - with a combination of type, user, state and the user-defined filter.
- The getAllAdocEvents Application Program Interface returns a list of adaptive documents and for each adaptive document all of its associated events. For example, all the services exposed by the adaptive document given the state of the adaptive document and the associated Activity Controllers. Each event is returned as an EventDetails object containing the event and the associated parameters.
- The getArchivedAdocs Application Program Interface returns a vector of AdocDetails for archived adaptive documents given a certain adaptive document type and the start and end date strings between which time the adaptive document was archived. The date format used is: `[MM: DD: YY] HH:MM AM/PM`.

Process Broker Services messaging clients

Any endpoint can communicate with Process Broker Services using MQ as the transport for guaranteed message delivery and Java Messaging Service as the messaging protocol. The messaging client interaction pattern shows how the messages (regardless of their type) are received and handled by Process Broker Services.

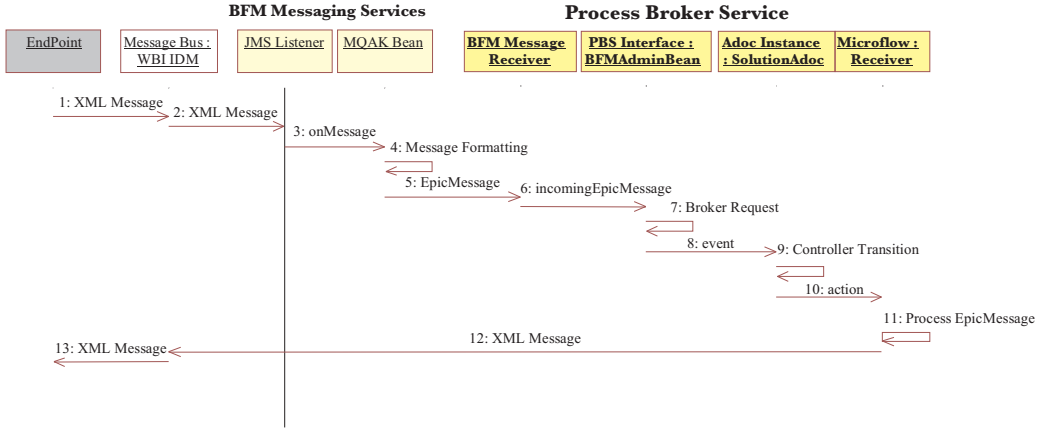

Figure 11. Process Broker Services Messaging Client Interaction Pattern

1. An endpoint generates an XML message. Typically an MQseries Adapter co-located with the endpoint, transforms the data from the endpoint to Open Application Group Business Object Document XML format. See http://www.openapplicationsgroup.com for OAG-BOD specifications.
2. WebSphere Business Integrator Information Delivery Manager, which serves as the message bus, routes the message to Business Flow Manager based on the content in the message header. The message bus also does any necessary message transformation. The Java Message Services Listener configured in the WebSphere Application Server on the Process Broker Services Node, picks up the message from the Process Broker Services Input Queue (BFMAIQ).
3. The onMessage method on the Java Message Services Listener invokes the MQseries Adapter Kernel Bean (this bean spawns worker threads as needed).
4. The message is formatted as required by the MQseries Adapter Kernel Bean using appropriate formatter classes.
5. The MQseries Adapter Kernel Bean then launches the Business Flow Manager Message Receiver (implemented as a stateless session Enterprise Bean) passing it the EpicMessage (this is the internal message wire format that is recognized by WebSphere Business Integrator components

using MQseries). The Business Flow Manager Message Receiver is the same for all message types that are received. It is registered as the ″command″ class for messages received on the BFMAIQ input queue.

6. Upon receiving the EpicMessage, the Business Flow Manager Message Receiver invokes the Process Broker Services Application Program Interface incomingEpicMessage (EpicMessage em) passing the EpicMessage as a parameter. This invocation is equivalent to a service request that a web client makes on Process Broker Services.
7. The Process Broker Services Interface then brokers the request. The brokering logic is based on extracting the following parameters from the message header:
   - BodyCategory (BC)
   - BodyType (BT)
   - BodySecondaryType (BST)
   - CorrelationId

   The Process Broker Services then constructs a message-mask containing <BC>.<BT>.<BST>. For example, consider a message with a bodycategory field of *EMP*, a bodytype field of *LoadReceiveable.SubmitBOD″* and a bodysecondary typefield of *003*. The message-mask for the example is then EMP.LoadReceivable.SubmitBOD.003. This message-mask is then used to lookup the adaptive document Type and the event to be raised. All message-masks must be registered in the bfm.properties file. For the above example, the bfm.properties entry is as follows:

   ```
   EMP.LoadReceiveable.SubmitBOD.003 = Registration, Submit
   ```

   This implies that the incoming EpicMessage is used to raise a Submit event on a Registration adaptive document Type.

   Note: The BodySecondaryType (BST) field of the EpicMessage is taken to be optional when constructing the message mask to correlate to the Adoc type or event. If the field is not specified (a null is encountered), it is ignored when constructing the message mask. In the this case, the message mask only consists of the bodycategory and bodytype fields.

   The Correlation Id in the message header is then used to identify the actual adaptive document instance, for example, the Correlation Id is equivalent to the adocId. If the Correlation Id is null then a new adaptive document is created and the adocId of the new adaptive document is then set as the Correlation Id of the message.
8. A service request is then made on the specific adaptive document instance, as identified in step 7, raising the event and passing in the following input parameters:

   MESSAGE
   the value is the entire EpicMessage

MESSAGEBODYCATEGORY
the value is BodyCategory field in the message

MESSAGEBODYTYPE
the value is the BodyType field in the message

ADOCREFERENCE
the value is the adocId

9. The event triggers a transition in the controller (Adaptive Document or Activity Controllers associated with the adaptive document).
10. As part of the transition, one or more actions are triggered. The action corresponds to a method invocation on a microflow built using the Flow Composition Builder tool (see *WebSphere Studio Business Integrator Extensions Developer's Guide* for details on using this tool). The microflow is registered as a command in the command.xml file and is invoked as an action from the controller.
11. The microflow processes the message, for example, it parses the message and operates on the content as appropriate. The operations could include creating new business objects, updating existing ones, or composing new messages to be sent (see Figure 12).
12. The microflow uses the source command classes to send a message using MQseries Adapter Kernel to the message bus.
13. The message is routed to the appropriate endpoint.

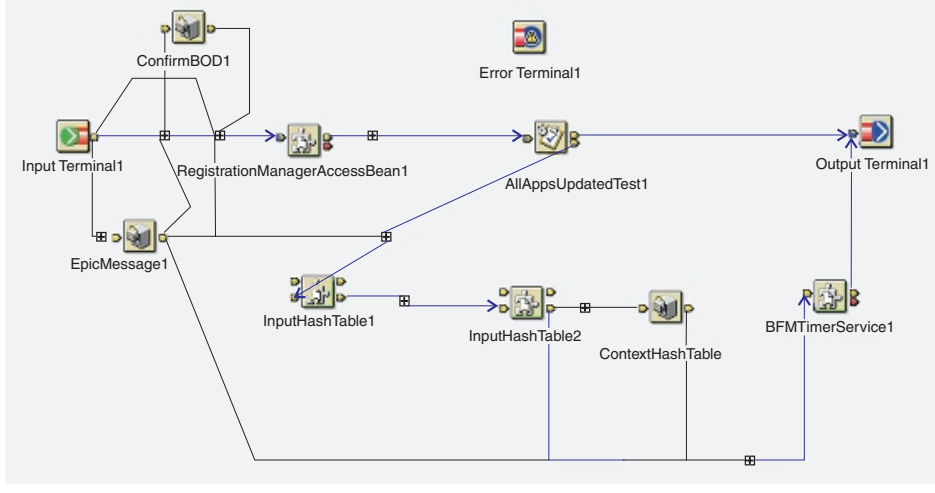

Figure 12. Example User Registration Microflow showing how Confirm BOD is processed

Handling automatic activities in workflows

The workflow engine can be viewed as a special client to Process Broker Services. See the *WebSphere Business Integrator Process Broker Services Concepts Guide* for details on how the workflow engine interacts with Process Broker Services at a component level. The details of handling automatic activities (for example, activities executed by some system component as opposed to humans) in a workflow using Process Broker Services are described here.

1. The HANDLEAUTOACTIVITY flag in the bfm.properties file must be set for Process Broker Services to handle automatic activities. If this flag is either set to NO, or not set at all, then Process Broker Services does not handle any automated activity.

   ```
   HANDLEAUTOACTIVITY = YES
   ```

2. When an activity becomes active, known to Process Broker Services through workflow events from WebSphere Workflow Services, Process Broker Services schedules a service request for execution either immediately, or at a specified time. The Process Broker Services scheduler service then launches either the default action-listener (also known as DefaulTaskServiceRequestHandler) or a user-defined action-listener (see "Chapter 4. Advanced Topics" on page 47 for writing user-defined action-listeners). These action-listeners or handlers then make the service request to raise the AutoExecute event on the associated Activity Controller.
3. The action-listeners or handlers must be registered with Process Broker Services for each automatic activity. The following entry in the bfm.properties file handles the registration.

   ```
   <Activity Name>HANDLER =<Handler Class>,<Run after/at>,<Num Retries>
   ```

   where:

   - Activity Name is replaced by the name of the automatic activity. Handler Class is a mandatory parameter and specifies the action-listener class. It is either a fully qualified name of the action-listener class or the keyword DEFAULT.
   - *Run after/at* is the parameter that specifies when the service request must start. The formats for specifying this include: [MM/DD/YY HH:MM AM/PM] to schedule on a specific date and time, [HH:MM AM/PM] to schedule at a specific time on the current day, and a fixed value such as 10 000 to schedule the request after 10 000 milliseconds. If this field is not specified, or does not conform to the mentioned formats, Process Broker Services starts the service request immediately.
   - The *Num Retries* parameter indicates the number of times the Process Broker Services Scheduler Service will retry the service request in case of failure.
4. The action-listener launches the service request with the following parameters that are available by default in the input and context Hashtable

of the service request:

Name	Value	Hashtable
User	Activity Owner	Input
ADOCREFERENCE	AdocId	Input
ACTIVITYID	Activity Identifier	Input
ROLES	Role of User	Context
ACTIVITYNAME	Name of Activity	Input
PROCDEF	Process Definition Name	Input
ACTIVITYVARS	Activity Variables	Input
INSTANCENAME	Activity Instance ID	Input

5. An Activity Controller when launched is always initialized to be in the Available State. For automatic activities, the Activity Controller must define a transition from the Available State with the AutoExecute event. This event is automatically triggered when the action-listener registered with Process Broker Services for automatic activity makes a service request. Note that the Activity Controller for automatic activities can have many other states and transitions – only the AutoExecute event transition is automatically started. An example of an activity controller with an AutoExecute event is shown below.

```
<statemachine id = "sm0">
<name>ReivewPO</name>
<state>
<name>Available</name>
<type>Normal</type>
<transition>
<target>Complete</target>
<event>ReviewPO.AutoExecute</event>
<condition></condition>
<action>Action 1</action>
... other actions ...
<action>WWFS_ForceFinish</action>
</transition>
</state>
</statemachine>
```

Chapter 4. Advanced Topics

This chapter describes advanced topics related to Process Broker Services solution development in the following sections.

Life-cycle management of adaptive documents

Life-cycle management of adaptive documents refers to the archival, revival, and removal of adaptive documents. The adaptive document behavior over its life-cycle is mainly governed by the Adaptive Document Controller definition. The Process Broker Services container also influences the life-cycle behavior of adaptive documents through some automatic life-cycle actions based on the state of the adaptive document.

Archiving adaptive documents

An adaptive document is archived under two conditions:

1.
```
<statemachine id = "TestAdoc">
<name>TestAdoc</name>
<state>
<name>Open</name>
<type>Normal</type>
<transition>
<target>TimeoutError</target>
<event>Timeout</event>
<condition></condition>
```

```
</transition>
</state>
<state>
<name>TimeoutError</name>
<type>Archive</type>
</state>
</statemachine>
```

2. When the adaptive document enters a state from which there are no defined transitions. In this case Process Broker Services automatically schedules the adaptive document for archiving. The default schedule for archiving is Immediate, but altering the following entry in the bfm.properties file can modify this default behavior: `<AdocType>ARCHIVETIME = [MM/DD/YY] HH:MM AM/PM or nnnnn milliseconds`, where the AdocType must be replaced by the actual type of the adaptive document.

   The AdocArchivalHandler, a System ActionListener provided by Process Broker Services in WebSphere Business Integrator Version 2.1 and registered with the Process Broker Services Scheduler Service, launches the archival process. The archival process involves the following steps:

   - Invokes an archive method on the Solution adaptive document. Any Solution adaptive document must implement this method to persist any solution attributes such as the business object references. If this method is not implemented, only the basic adaptive document attributes in AdocDetails are persisted.

     A sample implementation of an archive method is shown below. The business object attributes are contained in the Hashtable as name, value pairs.

     ```
     public Hashtable archive () throws java.rmi.RemoteException {
         try {
              Hashtable params = new Hashtable();
              params.put("BOID", getBoId());
              return params;
         } catch (Throwable e) {
              throw new RemoteException(e.getMessage());
         }
     }
     ```

   - Creates an instance of ArchivedAdoc with the solution and the base adaptive document attributes.
   - Removes the Solution adaptive document.

Reviving adaptive documents

Archived adaptive documents are revived using the Application Program Interface, Object reviveAdoc(String adocId), or an adaptive document to revive is selected using Vector getArchivedAdocs (adocType, StartDate, EndDate). See “Appendix B. Process Broker Services Application Program Interfaces” on page 69.

An automatic mechanism is not provided in Process Broker Services to revive adaptive documents. The Solution Developer must implement a utility using Process Broker Services Application Program Interfaces to revive adaptive documents. Also, if a Solution adaptive document has to be revived, the adaptive document must implement the method public void ejbCreate (java.lang.String argAdocId). A sample implementation of this method is provided below (**Attention: It may not work exactly this way in all situations**).

```
public void ejbCreate(java.lang.String argAdocId) throws javax.ejb.CreateException,
java.rmi.RemoteException
{
_initLinks();
// All CMP fields should be initialized here.
try {
adocId = argAdocId;
if (_DEBUG)
System.out.println("TestAdoc.ejbCreate(String) - creating TestAdoc with id " +
adocId + ". Creating the Base Adoc");
// create the base adoc
super.createAdoc(adocId, "Test");
if (_DEBUG)
System.out.println("Test Adoc created ..");
} catch (Throwable e) {
throw new RemoteException(e.getMessage());
}
}
```

Removing adaptive documents

An adaptive document is scheduled for removal when it enters a state of type Terminal. See the sample below. If there are transitions defined from such a state, they are ignored by Process Broker Services.

```
<statemachine id= "TestAdoc">
<name>TestAdoc</name>
<state>
<name>Open</name>
<type>Normal</type>
<transition>
<target>TimeoutError</target>
<event>Timeout</event>
<condition></condition>
</state>
<state>
<name>TimeoutError</name>
<type>Terminal</type>
</state>
</statemachine>
```

When an adaptive document is scheduled for removal, a removal task is automatically scheduled on the Process Broker Services Scheduler. The AdocRemovalHandler, a System ActionListener provided by Process Broker Services in WebSphere Business Integrator Version 2.1 and registered with the

Process Broker Services Scheduler Service, executes the removal. The default schedule for removal is Immediate, but altering the entry in the bfm.properties file can change this default behavior. Edit the following entry in the properties file:<AdocType>REMOVETIME = [MM/DD/YY] HH:MM AM/PM or nnnnn milliseconds, where the AdocType must be replaced by the actual type of the adaptive document. This implies that either the adaptive document can be scheduled to be removed at a certain date and time, or it can be removed after a certain time interval after its enters the Terminal state.

Process Broker Services scheduler

The Process Broker Services Scheduler enables Process Broker Services to schedule service requests for execution at a given time. This capability allows Process Broker Services to model asynchronous behavior. Following are some of the ways that the Process Broker Services Scheduler is used:

- Consider a scenario where a timeout has to be modeled in an adaptive document. If a request is not received within a certain time, raise a timeout event on the adaptive document. This is accomplished by scheduling a service request on the Process Broker Services Scheduler to raise a Timeout event on the adaptive document after a given time (See "Process brokering patterns" on page 52).
- Process Broker Services handles automatic Activities in a workflow by using the Process Broker Services Scheduler (See "Handling automatic activities in workflows" on page 44).
- It is possible to cascade service requests (for example, service requests that are executed one after another, but not in the same unit of work), by scheduling a service request from within a transition that is triggered by another service request. (See "Process brokering patterns" on page 52).

Process Broker Services provides a number of system commands for scheduling service requests. See "Process Broker Services system commands" on page 21 for definitions of these commands. Tasks that are scheduled on the Process Broker Services Scheduler Service are executed by Action Listeners (also referred to as Handlers). Process Broker Services provides system Action Listeners such as the AdocArchivalHandler for archiving adaptive documents and AdocRemovalHandler for removing adaptive documents. It is also possible to define custom Action Listeners or Handlers.

Writing custom action listeners or handlers

Use the following steps to write a Custom Action Listener for use in the Process Broker Services Scheduler.

1. Import the SDK bfm.sdk.zip into your development environment.
2. Extend the com.ibm.epic.bfm.timer.DefaultGenericServiceRequestHandler.

3. The Custom Action Listener must have a constructor like the example shown below.

```
public MyHandlerConstructor (String namedEvent,
java.util.Hashtable context,
java.util.Hashtable input,
String refId,
String user,
com.ibm.epic.bfm.ejb.BFMAdmin bfm)
{
super(namedEvent, context, input, refId, user, bfm);
}
```

4. The Custom Action Listener can override the doServiceRequest Process Broker Services Application Program Interface to add the custom code to augment for instance, the input and context Hashtable. Essentially, all the fields are protected variables in the base class. They can be accessed by directly referencing them as, *_namedEvent*, *_context*, *_refId*, and *_bfm*. A sample code fragment for a custom Action Listener is shown below.

```
public void doServiceRequest () {
// add/modify the service request parameters
_context.put ("MYCONTEXTATTRIBUTE&", value);
// other attributes
_input.put ("MYINPUTATTRIBUTE&", value);
// other attributes
// call the service request method in the base class
super.doAdocServiceRequest();
}
```

5. Ensure that the Custom Action Listener run time classes are in the class-path of the Process Broker Services Scheduler Service Dispatcher.

Process Broker Services scheduler service dispatcher

The Process Broker Services Scheduler Service Dispatcher (also known as the Timer Dispatcher) is installed as a Service on Windows NT® (in WebSphere Business Integrator Version 2.1) and is started when the Process Broker Services node boots. The service daemon polls for scheduled service request entries using the TimerService Enterprise Beans. The polling interval can be modified with the following entry in the bfm.properties file:`TIMERPOLLINTERVAL = 10000 // unit is milliseconds.`

The Process Broker Services Scheduler Service Dispatcher launches individual threads for each of the timer entries that are ready to be started. Each thread makes the appropriate serviceRequest calls back to Process Broker Services with the parameters established when the request was scheduled. To launch a number of threads concurrently, irrespective of the number of active timer entries, the following property in the bfm.properties file can be specified.
`MAXTHREADS = 50`

An Action Listener must be specified when scheduling the request. A default Action Listener is provided that makes the service request using the data (input, context) provided with the scheduled request.

Each scheduled request is launched on a different thread. Error trace can be turned on by setting the flag TIMERDISPATCHERDEBUG = YES in the bfm.properties file. Only error messages are written out to files in the working directory specified above. The file name format is: `tt.<bfmRefId>.err`

bfmRefId is the Adoc/Task Id for which the service request was scheduled.

To view trace output, the Windows NT service entry property for *Allow Service To Interact With Desktop* needs to be checked, to have trace output displayed on the console.

Process brokering patterns

Process Brokering Patterns encapsulate the design experiences with adaptive documents. They can be viewed as building blocks in the formation of a solution model using Process Broker Services. A Process Brokering Pattern is not specific to a solution, but it is based on scenarios that are commonly found in business process integration and management problems.

Each of the following patterns is described in terms of the modeling scenario, the solution, and the usage of the pattern.

Event with Time Window

Modeling Scenario

A Solution adaptive document enters a certain state and a business event associated with that state must occur within a given time window. If the event times out, then a suitable recovery action must be launched. The Event with Time Window Process Brokering Pattern handles this modeling scenario.

Solution

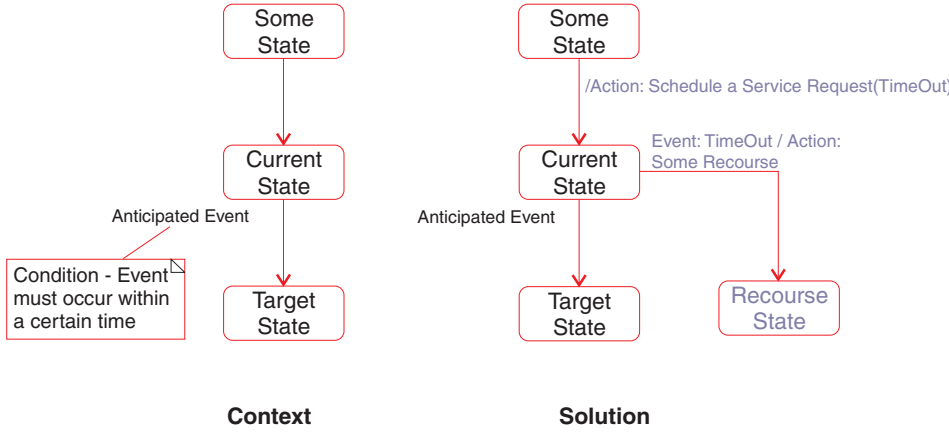

Figure 13. Event with Time Window Pattern

The state chart on the left in Figure 13 describes the context for the modeling pattern. The adaptive document is in the current state and the anticipated event has to occur within a certain time window.

The state chart on the right in Figure 13 describes the brokering pattern:

1. In the transition that placed the adaptive document in the current state, add a command that schedules a service request with the Process Broker Services Scheduler with the Time Window of the anticipated event.
2. Once the timeout has occurred, the Process Broker Services scheduler raises the TimeOut event.
3. The appropriate recovery is modeled based on the TimeOut event, such as, rolling back to the previous state, a self-transition on the current state, or moving to a new state. During any of these transitions appropriate actions can be taken for the recovery.

Usage

- This pattern is used for cascading service requests on the same adaptive document instance, for example, triggering one service request followed by another where one triggers the other, although not in the same unit-of-work. This is done by making the TimeOut factor Immediate on the Process Broker Services Scheduler.
- This pattern also illustrates how the adaptive document can model asynchronous behavior, meaning the scheduling of service requests within a transition that in turn triggers another transition asynchronously to execute actions.
- The TimeOut scenario is common in situations, such as an acknowledgement from an endpoint application following a message that

was sent a priority. If no acknowledgement is received within a given time, then suitable recovery actions need to be executed.

Non-deterministic conditional

Modeling scenario

When a business event is raised against a Solution adaptive document, the behavior is non-deterministic in that the same event can transition the adaptive document to one among many other possible target states based on mutually exclusive conditional business logic.

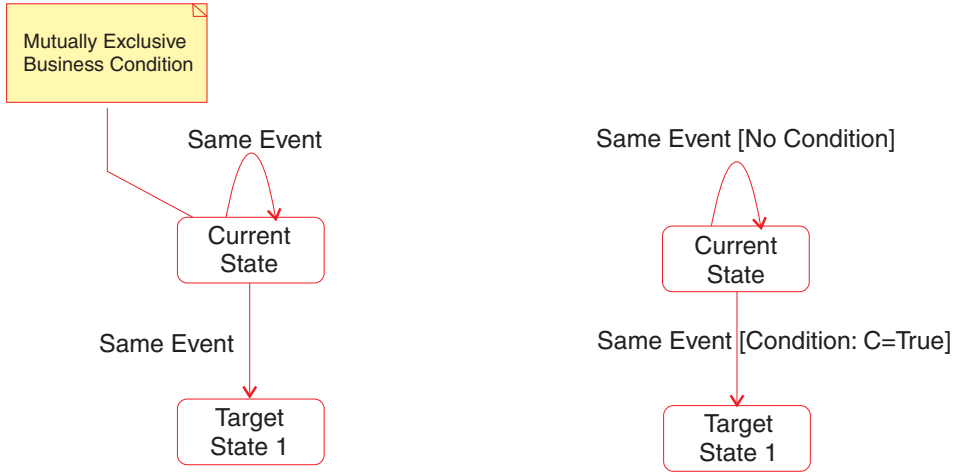

Figure 14. Non-Deterministic Conditional Pattern

Solution

The state chart on the left in Figure 14 explains the context of this pattern. The same event can trigger a self-transition of the adaptive document in the current state as well as transition it to Target State 1 (there could be many other possible states). The state machine is therefore non-deterministic.

The state chart in the right of Figure 14 reveals the non-deterministic conditional pattern.

- The solution involves using conditional logic on the transitions where the conditional logic expressed in the set of transitions is mutually exclusive.
- The transition Current State→Target State 1 is triggered when the conditional expression evaluates to true. The self-transition does not have any conditional logic defined. The transitions are ordered in the controller definition such that the self-transition follows the transition to Target State 1.
- Process Broker Services evaluates the transitions for an event from a given state in a controller in the order in which they appear in the controller

definition, triggers the transition that first evaluates as true, and skips the rest. If the condition evaluates to false, then the transition to Target State 1 does not occur and the self-transition is triggered. Otherwise, the first transition occurs and the second transition is skipped.

Usage

This pattern is useful in modeling situations where the adaptive document receives confirmation messages from several endpoints. It needs to be in the same state until it has received a certain number of confirmations (n), upon which it makes the transition out of the state. In this case, a business object can be used to keep track of the number of confirmations that have been received. The conditional expression on the transition out of the current state can be defined so that it queries the business object and returns true if n–1 confirmations have already been received. Further, the self-transition can involve an action that calls the same business object, to just update the counter every time a confirmation is received unless it has received the maximum number of confirmations.

Activity Mediator

Scenario

A Solution adaptive document (for example, PO adaptive document) can transition from a given document state, such as To-Be-Approved, to a specific document state, such as Approved, only when two independent activities are completed. Further, the two activities can complete in any order. This could be activities that are either executed in parallel within a process, or belong to two separate processes, for example, Technical Approval and Financial Approval are complete.

Solution

The context and solution to the scenario is shown in Figure 15 on page 56. The scenario calls for the coordination of the adaptive document life-cycle with that of two independent activities that can be part of different business processes and can engage different systems and people.

The solution uses the mediation capabilities of the Process Broker, where the Financial Approval and the Technical Approval activities execute independently. The completion events in either controller are also raised against the adaptive document controller. Process Broker Services executes all transitions that are eligible to be triggered for a given event across controllers in an adaptive document, in a single unit-of-work. This implies that when the FinancialApproval.Complete event is raised and suppose the adaptive document is in the Technical Approval Done state, then the transitions in the Financial Approval controller as well as the transition to the Approved state in the Adaptive Document Controller are executed in a single unit-of-work. It can be seen that the activity mediation (for example, activities loosely coupled and indirectly coordinated through the adaptive document) ensures that

regardless of the order in which the activities are completed the adaptive document moves to the Approved state only when the Financial Approval and Technical Approval are complete.

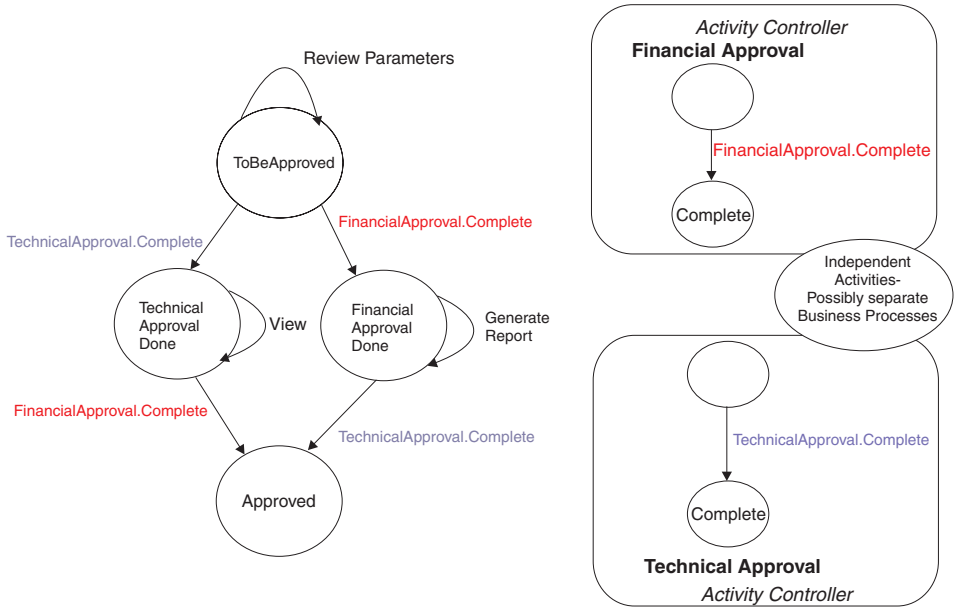

Figure 15. Activity Mediator Pattern

Usage

The Activity Mediation is a common Process Brokering pattern in business process integration problems where multiple business processes have to be integrated to provide an end-to-end business control and view.

Web of responsibility

Scenario

When a Solution adaptive document, such as an RFQAdoc enters a specific state, such as Cancelled for a certain business event, the web of activities associated with the adaptive document need to be appropriately updated. These activities are essentially independent activities that are associated with a single adaptive document.

Solution

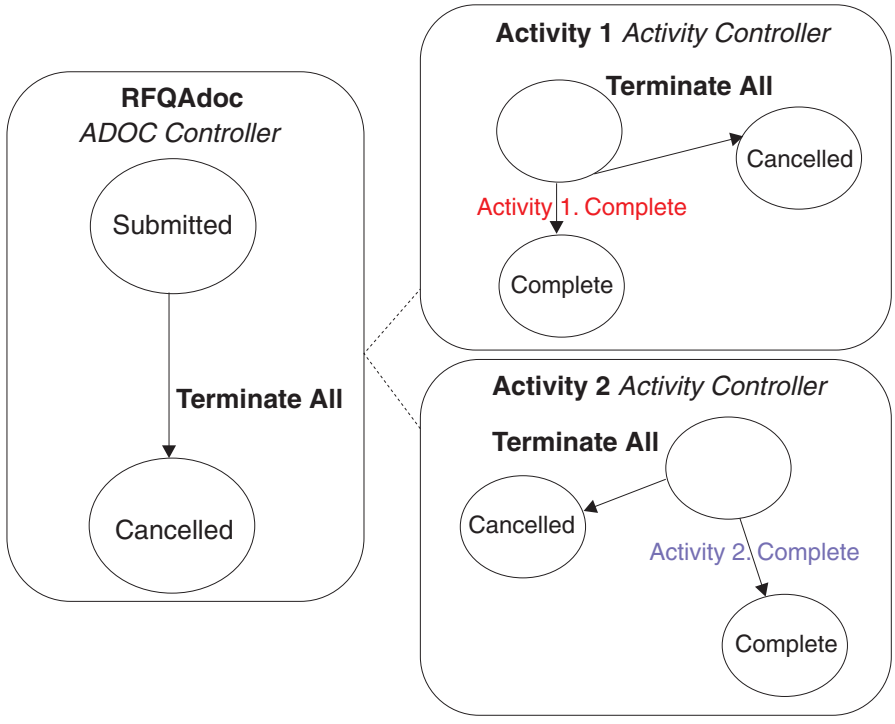

Figure 16. Web of Responsibility Process Broker Pattern

The context and solution for the scenario is shown in Figure 16. When the RFQAdoc is cancelled the web of activities in which the RFQAdoc is involved also need to be cancelled. This is accomplished by raising the same event on the web of activity controllers. Process Broker Services executes the transition for a given event across all controllers for an adaptive document within the same unit-of-work.

Usage

The Web of Responsibility pattern is common in business process brokering where there are hierarchical dependencies between business entities, such as between a Purchase Order and its line items coming from different vendors, participating in separate business processes resulting in a web of responsibility for the root entity. For example, each line item may be involved in an individual shipping process that in turn engages different logistics participants.

Dynamic Collaboration

Scenario

The Solution adaptive document spawns and participates in a long-running business process where one of the activities in the long-running process in turn generates a dynamic set of child activities that are executed in parallel. The number of child activities is dynamically determined by some business logic. Further, the parent activity must be completed only after the conclusion (completion or timeout) of the child activities.

Solution

The context and solution for the Dynamic Collaboration Broker Pattern is

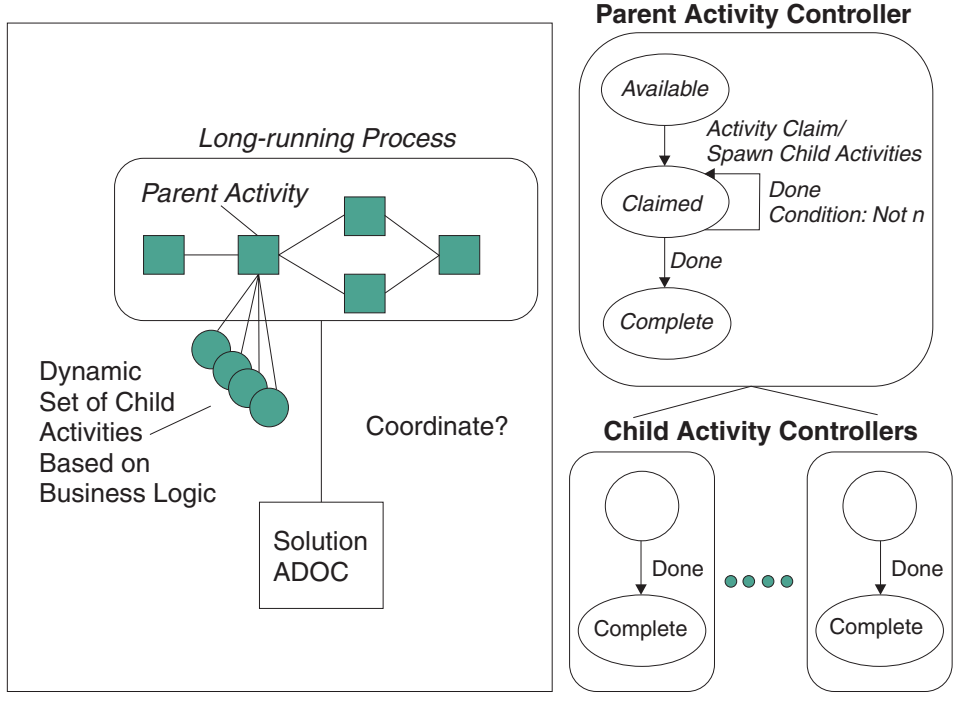

Figure 17. Dynamic Collaboration Process Broker Pattern

shown in Figure 17. The Solution adaptive document spawns a long-running business process. One of the activities in the process, referred to as the Parent Activity, in turn spawns a number of Child Activities where the number is computed dynamically. The Parent Activity can complete only after all the Child Activities are completed.

The Solution involves spawning the dynamic set of activity controllers based on some business logic encapsulated in a command in the transition of the Parent Activity controller. Each of the Child Activity controllers raises the Done event upon completion. These same events are also raised on the Parent

Activity Controller. The Parent Activity Controller then uses the Non-Deterministic Conditional Pattern to coordinate the completion of the child activities; the conditional logic checks to see if all the child activities are completed, where the conditional evaluation can be encapsulated in user defined evaluate function (see "User defined conditional logic" on page 18.

Usage
The Dynamic Collaboration pattern is common in business problems that involve business processes spanning multiple parties. For example, consider a cascaded RFQ process where one of the activities is to match the customer submitting an RFQ with a set of vendors, determine the dynamic set of vendors, and initiate a number of child Quote processes. Following the receipt of all quotes, the Quote step is completed and the process moves forward. See the *WebSphere Business Integrator Process Broker Services Concepts Guide* for an example of the Dynamic Collaboration Pattern.

Assigning a user-generated adaptive document Identifier

Process Broker Services provides a helper method on the AdocProxy class that generates an unique id based on a given string and time stamp. The method is called in the ejbCreate() of the Solution adaptive documents, and is as follows: `String poadocId = super.getUniqueString ("PO");`

Modifying this method can generate unique adaptive document ids: `String poadocId = yourUUIDService.getUUId ("PO");`

The size limit for the adaptive document Id is 32 characters.

Dynamically incorporating changes to controllers

Process Broker Services can be configured to dynamically incorporate any changes in the controller, command, and receiver xml files without stopping the Process Broker Services Application Server. This feature is turned on and off through the ACCEPTFILECHANGE flag in the bfm.properties file: `ACCEPTFILECHANGE = YES`

Receiver caching

Receiver caching has two main aspects: caching the Receiver Home, and caching of Receiver instances within a unit-of-work. The receiver Home is cached in Process Broker Services and is not looked up with each invocation. This implies that when a receiver is restarted, the cache is likely to be outdated, resulting in a transaction rollback. However, in the case of Enterprise Bean receivers, the cache is refreshed automatically upon the next invocation, and the transaction rollback error is cleared.

In the case of caching Receiver instances in a unit-of-work, the Process Broker Services default behavior is to cache the Receiver instance after the first invocation. This implies that in a controller transition, if two or more commands target the same receiver, the same receiver instance is used in repeated invocations. To change this default behavior, such as spawning a new instance of the receiver for each command, a copy of the receiver definition must be placed in the receiver.xml file with a different receiverId. Subsequent commands can then use the copy of the receiver by having the command in the command.xml file refer to the new receiverId.

Overriding the DB2 userId and password for Process Broker Services Queries

The DB2 user id and password is read from LDAP under `dn="cn=db2admin,o=ePICUsers,o=epic",name=uid` and `dn="cn=db2admin,o=ePICUsers,o=epic",name=userPassword`. To override the default or if LDAP is unavailable, the values are read from the bfm.properties file. The uid with the name 'DBUser' is expected and the password is expected under the name 'DBPassword'.

Advanced tips

- Nested service requests are supported, for example, a service request can be called as a command from within another service request. However, such nested service requests cannot be made on the same adaptive document instance. This violates the Enterprise Bean re-entrant property and the result will be unpredictable, possibly resulting in a transaction rollback.
- Multiple Process Broker Services AppID: The application Name lookup in LDAP can be changed to start at a different AppID node, rather than defaulting to o=ePICAppID as *BFM* in LDAP. A new variable has been added to specify the Process Broker Services application name during deployment. Specify the following environment variable in the Application Server command line: `-Dbfm.ldapname=ACMEBFM`

Chapter 5. Troubleshooting

Process Broker Services runs as an Enterprise Bean application inside the container provided by the WebSphere Application Server. Some of the errors happening inside the container though caused by the Process Broker Services, do not necessarily appear at the client invocation layer. Troubleshooting a Process Broker Services error can pose challenges. Following is a step-by-step guideline to troubleshoot an error caused by a Process Broker Services application.

For best results, use the following Checklist sequentially; going to the next step if the current method does not detect the cause of the problem.

Checklist

The following conformance points must be checked:

- All Solution adaptive documents must be deployed in the same Application Server as Process Broker Services, although they are not required to be in the same container.
- All Solution adaptive documents are deployed with parameters as specified in "Building an adaptive document" on page 4. Important parameters to check are the Transaction and Read-only attributes on the Enterprise Bean remote interface methods.
- The Data-source defined for the Process Broker Services and Solution adaptive documents container needs to use the Java Transaction Application Program Interface (JTA) enabled driver from DB2. To verify, check the following items:
  - Confirm the following entries on the Database driver used for the data-source
    - Class Name is COM.ibm.db2.jdbc.app.DB2Driver
    - URL prefix is jdbc:jta:db2
    - JTA enabled is True
  - The DB2 JDBC driver (db2java.zip) must use the JDBC2.0 drivers. To confirm verify that:
    - The <DB2 install>\java12\inuse file is JDBC 2.0
    - <DB2 install>\java\db2java.zip files has *XA class files

Trace files

Process Broker Services pbsout.txt and pbserr.txt are trace output files. They are found in the <WBI Install>\logs directory. Depending on the trace/debug settings in the bfm.properties file (see "Appendix A. Process Broker Services properties" on page 65), the trace content can be controlled. These files provide information on the cause of the failure that needs to be investigated.

Operation specific errors

If the trace output files do not provide enough information about the cause of the error, then the error is most likely caused due to some run time exceptions that Process Broker Services was not able to handle. Some of the common run time errors are listed below, grouped by the nature of the operational tasks performed by Process Broker Services.

Creating an adaptive document

The following errors can happen during the creation or initialization of an adaptive document.

Note: Set the trace levels/flags appropriately to obtain trace output in the pbsout and pbserr files (See "Trace and debug flags" on page 65 for details about setting the trace/flags)

Link exception

This exception occurs if the client stubs generated for the Solution adaptive documents are not available in the Process Broker Services class-path. This is a requirement because Process Broker Services acts as a client to the Solution adaptive documents to instantiate them during run time and as such, needs the client jars to make the appropriate bindings.

To resolve this, extract the client stubs generated from the Solution adaptive documents into the class-path of Process Broker Services.

Controller initialization failed

This error in most cases is the result of a malformed controller/command/receiver XML file. Check pbserr.txt to see possible parsing errors and the location at which the error occurred. Correct the error and retry.

Transaction rollback

In this case, the container has thrown a transaction rollback exception and the trace output does not readily show the cause. This happens when a run time exception has occurred in the Enterprise Bean container code, and the error does not make it back to the application level. The only way to debug this is to take a trace dump on the Process Broker Services Application Server using the trace facilities provided in the WebSphere Application Server. The

mechanism to perform this operation is discussed in detail in WebSphere Application Server documentation, but the basic approach is described in "WebSphere Application Server trace dump" on page 64.

Making a service request

Service request transaction rollback

A transaction rollback is indicated in the call return or the trace files, but the cause is not identifiable. Perform the following to find the cause:

- The pbsout.txt file indicates the command that failed. The trace contains an entry that looks like the following: `COMMAND <CommandId> FAILED`
- Check the trace for the receiver that this commands executes to see if the error occurred in the receiver. If the receiver trace shows an error, fix it.
- The service request happens under a distributed transaction spawned by the Enterprise Bean container. As such all commands invoked in the transition need to be executed on receivers that support the Java Transaction Application Program Interface protocol. If they do not and the receiver is an Enterprise Bean, then the corresponding method name should be marked as TX_NOT_SUPPORTED.
- If the receiver is an Enterprise Bean receiver, check that the receiver client stubs are present in the Process Broker Services application server class-path.
- Use the Application Server trace dump to investigate further. Details about taking the trace dump are provided in "WebSphere Application Server trace dump" on page 64.

Process Broker Services scheduler errors

There is a retry count for every request scheduled using the Process Broker Services Scheduler. On expiration of the entry, Process Broker Services uses the registered Action Listener/Handler to make the appropriate call back to Process Broker Services. These calls are either calls to make a service request or to create, archive, or remove an adaptive document. The TimerDispatcher decrements the retry count every time it fails in the call. When the retry count is zero, the timer entry is assumed to be not able to be processed and no further action is taken. To resolve the reason for such a timer entry the following steps can be taken:

- If the scheduler dispatcher trace/debug flag is turned ON (See "Trace and debug flags" on page 65), the dispatcher generates a trace file for each timer entry.

  Note: If the call back invocation for the entry is successful, there is no entry. The file name convention used is `tt.<AdocId>.err`
- If the scheduled event is the removal or archival of an adaptive document, make sure the retry count is set to a high number (for example, 20). If concurrent removal or archival requests are placed to Process Broker

Services, transaction rollbacks happen due to timeout or deadlocks. In Process Broker Services in WebSphere Business Integrator Version 2.1a high retry count is recommended for removal and archival schedule requests.

WebSphere Application Server trace dump

To take a WebSphere Application Server trace dump, perform the following:

1. Open the WebSphere Application Server console for the Process Broker Services WebSphere Application Server node (same as the Business Flow Manager node).
2. Right-click on the Process Broker Services Application Server, while it is running, and select **Trace** to view the Trace Administration window.
3. In the Window, right-click on Components and select **All**. Then select **Set**.
4. Select **Apply** on the WebSphere Application Server Console.
5. Run the application transaction that was causing the failure.
6. After the error occurs, right-click on the Application Server node on the Console to bring up the Trace Administration window .
7. Provide a file name for the `Dump File Name`, where you want the WebSphere Application Server to write the trace, for example <WBI Install>\logs\pbswasdump.txt
8. Select **Dump** and the trace is written to the file.
9. Open the dump file with an editor and scan for errors. It is normally a large file, so it may be advisable to search for the word ***Exception*** to go to the line in the dump where the exception is recorded.

Appendix A. Process Broker Services properties

Process Broker Services uses the bfm.properties file to store and read run time properties. Following is the list of properties that Process Broker Services expects to be available in the bfm.properties file.

- Package names
- Controller file locations
- Trace and debug flags
- Process Broker Services scheduler dispatcher
- Automated activities
- Miscellaneous.

Package names

Process Broker Services run time uses its own utility classes to lookup the JNDI Homes for Enterprise Beans that are directly accessed within Process Broker Services. The Solution adaptive documents fall into this category. The format for the entry is: `<AdocName>Package=<PackageName>`. For example:

```
TestAdocPackage = com.ibm.epic.bfm.samples
POAdocPackage = com.ibm.epic.acme.bfm
```

Refer to “Building an adaptive document” on page 4 for details.

Controller file locations

The mapping between the controller file names and their physical location on the file system is expected to be in the bfm.properties file. The format is: `<Adoc/Activity>ControllerXML = <file system path>/<file name>`. For example:

```
TestAdocControllerXML = D:/../xml/testAdocController.xml
```

Refer to “Creating and configuring controllers” on page 10 for details.

Trace and debug flags

Tracing output for different parts of Process Broker Services is turned on or off using these flags. Trace output is written to the files that have been specified in the Application Server’s stdout and stdin configuration properties. By default they are named pbsout.txt and pbserr.txt

The format of an entry is: `<PBSComponent>Debug = YES/NO`

Process Broker Services properties

The following flags are used to collect trace information for different parts of the Process Broker Services run time.

BFMAdminDebug
This flag traces the different client invocations on the Process Broker Services.

AdocDebug
This flag traces the lifecycle events on the Adaptive Document such as create, archive, remove, and restore.

ControllerDebug
This flag traces the activities within the controller engine: the state machine, the commands invoked, and how they are being executed on the receivers.

BaseDebug
This flag traces the miscellaneous run time utilities classes available in the Process Broker Services runtime using the JNDI home lookups.

<Solution>AdocDebug
This flag outputs a trace from the Solution adaptive documents.

PUBLISH_EVENTS
This flag controls whether Process Broker Services publishes workflow events, such as the availability of an activity, an activity being claimed, and so on. For Process Broker Services in WebSphere Business Integrator Version 2.1 it is recommended to set flag to NO.

Process Broker Services scheduler dispatcher properties

The Process Broker Services Dispatcher uses the following flags. Their values can impact the load on Process Broker Services. Therefore care must be exercised in their use.

TIMERPOLLINTERVAL
The format is, TIMERPOLLINTERVAL =<nnnnnn> milliseconds. The Process Broker Services Scheduler Dispatcher checks every nnnnnn milliseconds for scheduled entries that have expired and need to be handled.

MAXTIMERTHREADS
The format is, MAXTIMERTHREADS =<n>. This number controls the maximum number of threads that the Process Broker Services Scheduler Dispatcher spawns to handle the active events, such as, events that are ready to be launched. One thread is launched per scheduled entry. For example, ten scheduled events need to be handled, but the maxtimerthreads is set to 5. In this case, only the first five entries for action listeners are launched in individual threads.

TIMERDISPATCHERDEBUG

This flag can be turned on or off to collect trace output from the Dispatcher and the different Action Listeners that are launched. The dispatcher generates a trace file for each timer entry.

Note: If the call back invocation for the entry is successful, there will be no entry. The file name convention used is `tt.<AdocId>.err`. Refer to "Process Broker Services scheduler" on page 50 for details.

Automated activities

Please refer to "Handling automatic activities in workflows" on page 44 for details.

Miscellaneous

ADOCSTRLEN

Process Broker Services uses the value specified in this entry to determine the number of characters to prepend to the generated adoc id. Suppose this value is set to **6** and the name of the adaptive document passed during creation is **SOLUTION**. The id generated by Process Broker Services is `SOLUTI<the currentTimeStamp>`. Making the value **3** generates an id of `SOL<thecurrentTimeStamp>`. Refer to "Building an adaptive document" on page 4 for details about the adaptive document name.

ACCEPTFILECHANGE

Set this flag to **yes** to prompt Process Broker Services to automatically refresh the internal controller definition cache if the XML files controller, command, or receiver are changed after Process Broker Services has started. The changes are read and the internal objects are refreshed without the need to restart Process Broker Services. Set this flag to **no** to disable this function.

Process Broker Services properties

Appendix B. Process Broker Services Application Program Interfaces

BFMAdminBean Class

```
java.lang.Object
  |
  +--com.ibm.epic.bfm.ejb.BFMAdminBean
```

public class BFMAdminBean
extends java.lang.Object and impletments javax.ejb.SessionBean

BFMAdmin is a stateless session bean that implements the interface provided by Process Broker Services. All methods in this bean are executed within a transactional context.

archiveAdoc

```
public void archiveAdoc(java.lang.String adocId)
                throws java.rmi.RemoteException
```

Archives the adaptive document. Archives the adaptive document specified by the *adocId* and any business attributes specified by the Hashtable archive() method in the Solution adaptive document. Normally this method is invoked automatically by the adaptive document Archival Listener when the archival request is scheduled. An adaptive document can also be scheduled for archival using the scheduling services. For more information please refer to "Life-cycle management of adaptive documents" on page 47.

Parameters:

adocId
The adaptive document identifier.

Throws:

java.rmi.RemoteException
Wrapped around any thrown exception.

createAdoc

```
public java.lang.Object createAdoc(java.lang.String adocType)
                           throws java.rmi.RemoteException
```

Creates an instance of an adaptive document for the given type.

Parameters:

adocType
The adaptive document type.

Returns:

Object
The solution adaptive document object. This must be *narrowed* before it is used.

Throws:

java.rmi.RemoteException
Wrapped around any thrown exception.

createAdoc

```
public java.lang.Object createAdoc(java.lang.String adocType,
                                   java.lang.String adocId)
                            throws java.rmi.RemoteException
```

This method is used to create an instance of the solution adaptive document.

Parameters:

adocType
The adaptive document type.

adocId
The adaptive document Id.

Returns:
Object The solution adaptive document object.

Throws:

java.rmi.RemoteException
The exception description.

createAdocIdReturn

```
public java.lang.String createAdocIdReturn(java.lang.String adocType)
                                    throws java.rmi.RemoteException
```

Creates an adaptive document for a given adaptive document type, or creates an adaptive document for a given adaptive document type, but returns the adaptive document identifier instead of the remote object.

Parameters:

adocType
The adaptive document type.

Returns:

String
The adaptive document Id.

Throws:

java.rmi.RemoteException
Wrapped around any thrown exception.

ejbActivate

```
public void ejbActivate()
                            throws java.rmi.RemoteException
```

This method is called when the Enterprise Java Bean is activated. This method is specified by the interface javax.ejb.SessionBean.

Returns:
Void

Throws:

java.rmi.RemoteException
The exception description.

ejbCreate

```
public void ejbCreate()
                            throws java.ejb.CreateException
                                   java.rmi.RemoteException
```

This method creates the bean.

Returns:
Void

Throws:

java.ejb.CreateException
The exception description.

java.rmi.RemoteException
The exception description.

ejbPassivate

```
public void ejbPassivate()
                            throws java.rmi.RemoteException
```

This method is called when the Enterprise bean changes to the passivation state. Cached objects are released. This method is specified by ejbPassivate in the interface javax.ejb.SessionBean.

Returns:
Void

Throws:

java.rmi.RemoteException
The exception description.

ejbRemove

```
public void ejbRemove()
                            throws java.rmi.RemoteException
```

This method is called when the Enterprise bean is removed. All internal references are cleared and all database connections are closed. This method is specified by ejbRemove in the interface javax.ejb.SessionBean.

Returns:

Void

Throws:

java.rmi.RemoteException
The exception description.

getAdoc

```
public java.lang.Object getAdoc(java.lang.Object key,
                                java.lang.String adocType)
                         throws java.rmi.RemoteException
```

Gets an adaptive document, or gets the remote Solution adaptive document object using the key for the adocType. The key object passed in must be the key object for the Solution adaptive document, otherwise a run time exception is thrown.

Parameters:

key
The primary key of the particular solution adaptive document, for example, *RFQAdocKey*

adocType
The adaptive document type, for example a Request for Quote.

Returns:

Object
The adaptive document Object. It must be narrowed to the appropriate adaptive document.

Throws:

java.rmi.RemoteException
The exception description.

getAdocs

```
public java.util.Vector getAdocs(java.lang.String adoctype,
                                                 java.lang.String user,
                                                 java.lang.String state)
                    throws java.rmi.RemoteException
```

Gets a list of Adaptive documents that match some criterion. Gets a list of AdocDetails for a given adocType, a give user, a given state, or any combination of the three.

Parameters:

adocType
The adaptive document type.

user
The adaptive document owner.

state
The adaptive document state.

Returns:

Vector
A list of AdocDetails objects.

Throws:

java.rmi.RemoteException
The exception description.

getAdocsByFilter

```
public java.util.Vector getAdocsByFilter(java.lang.String adocType,
                                         java.lang.String user,
                                         java.lang.String state,
                                         java.util.Hashtable filterParams)
                                  throws java.rmi.RemoteException
```

Gets a list of adaptive documents based on a user filter. The filtering logic is implemented in the boolean filter (Hashtable filterParams) method on the solution adaptive document. The filter method returns a true or false, based on whether the adaptive document is selected or is not selected. This allows selection of an adaptive document based on referenced business object attributes.

A null can be passed for any or all of the adaptive document attributes: type, owner, or state.

Parameters:

adocType
The adaptive document type.

user
The adaptive document owner.

state
The adaptive document state.

Hashtable
Filter parameters as name-value pairs.

Returns:

Vector
A list of AdocDetails objects.

Throws:

java.rmi.RemoteException
The exception description.

getAllAdocEvents

```
public java.util.Vector getAllAdocEvents(java.lang.String adocType,
                                         java.lang.String state,
                                         java.lang.String role,
                                         java.lang.String user)
                                  throws java.rmi.RemoteException
```

Gets a list of adaptive documents and for each adaptive document a list of events that the adaptive document can accept in the current state. The event list includes the ones that can be raised on Activities with which the adaptive document might be associated. The event and associated attributes are wrapped in an EventDetails object. The input parameters can be provided in any combination, although a null must be specified if the particular attribute is insignificant.

Parameters:

adocType
An adaptive document type. Provide null if unknown.

state
An adaptive document state. Provide null if unknown.

role
The Role of the interacting user. Provide null if unknown.

user
The name of the user. Provide null if unknown.

Returns:

Vector
A Vector of AdocEvents.

Throws:

java.rmi.RemoteException
The exception description.

getAllPossibleBusinessEvents

```
public java.util.Vector getAllPossibleBusinessEvents(java.lang.String adocId,
                                                     java.lang.String role,
                                                     java.lang.String user)
                                              throws java.rmi.RemoteException
```

Gets a list of events that the adaptive document can accept in the current state. The event list includes the ones that can be raised on Activities with which the adaptive document might be associated. The event and associated attributes are wrapped in an EventDetails object.

Parameters:

adocId
An adaptive document Instance identifier.

role
The Role of the interacting user. Provide null if unknown.

user

The name of the user. Provide null if unknown.

Returns:

Vector
A Vector of named event String.

Throws:

java.rmi.RemoteException
The exception description.

getArchivedAdocs

```
public java.util.Vector getArchivedAdocs(java.lang.String adocType,
                                         java.lang.String startDateTimeStr,
                                         java.lang.String endDateTimeStr)
                                  throws java.rmi.RemoteException
```

Gets a list of adaptive documents that have been archived. This is probably a call that is made as the first step to revive or restore an adaptive document or a group of adaptive documents. The following adaptive documents are returned:

- All adaptive documents that have been archived within the given period of time (both start and end time is provided).
- All adaptive documents that have been archived before a particular time (only end time is provided).
- All adaptive documents that have been archived after a particular time (only start time is provided).

Parameters:

adocType
An adaptive document type. Provide null if unknown.

startDateTime
Provide in the format [MM:DD:YY] HH:MM AM/PM. For example, 4/12/01 1:40 AM, 1:00 PM (1:00 PM today) or null.

endDateTimeStr
Provide in the format [MM:DD:YY] HH:MM AM/PM. For example, 4/12/01 1:40 AM, 1:00 PM (1:00 PM today) or null.

Returns:

Vector
A Vector of AdocDetails.

Throws:

java.rmi.RemoteException
The exception description.

incomingEpicMessage

```
public java.util.Hashtable incomingEpicMessage
                              (com.ibm.epic.adapters.eak.mcs.EpicMessage em)
                                   throws java.rmi.RemoteException
```

This method provides the necessary logic to map or transform an incoming message (EpicMessage) to raise an event on an adaptive document instance using the serviceRequest method. For more information refer to "Process Broker Services messaging clients" on page 41.

Parameters:

EpicMessage
A full formed message as an EpicMessage.

Returns:

Hashtable
The output, similar to the ones produced by the serviceRequest method.

Throws:

java.rmi.RemoteException
The exception description.

incomingEpicMessage

```
public java.util.Hashtable incomingEpicMessage(java.lang.String bodyCategory,
                                               java.lang.String bodyType,
                                               java.lang.String corrId,
                                               java.lang.String message)
                                        throws java.rmi.RemoteException
```

This method provides the necessary logic to map/transform an incoming message (EpicMessage) to raise an event on an adaptive document instance using the serviceRequest method. For more information refer to "Process Broker Services messaging clients" on page 41.

Parameters:

bodyCategory
The body category of the message. This is mapped to an adaptive document type

bodyType
The body type of the message. This is mapped to the event to be raised on the adaptive document.

corrId
The correlation Id of the message. This is assumed to be the adaptive document Id.

message
The message itself in a string form.

Returns:

Hashtable
The output, similar to the ones produced by the serviceRequest method.

Throws:

java.rmi.RemoteException
The exception description.

initializeAdocDoServiceRequest

```
public java.util.Hashtable initializeAdocDoServiceRequest
                                                    (java.lang.String request,
                                                     java.util.Hashtable context,
                                                     java.util.Hashtable input,
                                                     java.lang.String adocType,
                                                     java.lang.String adocId,
                                                     java.lang.String user)
                                              throws java.rmi.RemoteException
```

This creates the adaptive document and then raises the event (request) as a service request. This method automatically creates the adaptive document and then makes a service request with the event (request) passed using the event parameters passed in the input and context hashtables. Normally the event is used to initialize the adaptive document, for example, create any BO's, assign the references to the adaptive document, set some BO attributes and so on. An example of an *Initialize* event is found in the testAdocController.xml file,

which is used for the adaptive document provided as the sample, TestAdoc. See "Service request to create and initialize an adaptive document" on page 38.

Parameters:

request
The event.

context
The context information.

input
The input information.

adocType
The type of adaptive document to be created. For example, PO or RFQ.

adocId
The adaptive document identifier to be used or null if Process Broker Services has to generate one.

user
The user invoking the operation. A null can be passed if the user is unknown.

Returns:

Hashtable
The output of adaptive document and task controller commands.

Throws:

java.rmi.RemoteException
The exception description.

invoke

```
public java.util.Hashtable invoke(java.lang.String commandId,
                                  java.util.Hashtable _context,
                                  java.util.Hashtable _inputList)
                           throws java.rmi.RemoteException
```

Allows execution of any command defined in the command.xml file without involving a controller.

Parameters:

commandId
The command Id, as defined in the command.xml file.

_context
Hashtable of context information.

_inputList
Hashtable of input information.

Returns:

Hashtable
Output values of the executed command.

Throws:

java.rmi.RemoteException
The exception description.

removeAdoc

```
public void removeAdoc(java.lang.String adocId)
                throws java.rmi.RemoteException
```

Removes the given adaptive document object. Life cycle management of adaptive documents are handled automatically. Use this method if required to handle removal of an adaptive document otherwise, use scheduleAdocRemoval on the TimerServices Enterprise Bean to schedule an adaptive document removal.

Parameters:

adocId
The primary key for the adaptive document object.

Returns:
void

Throws:

java.rmi.RemoteException
The exception description.

reviveAdoc

```
public java.lang.Object reviveAdoc(java.lang.String adocId)
                           throws java.rmi.RemoteException
```

Revives the adaptive document specified in the adocDetails. If a revive (Hashtable solutionAttributes) is implemented on the Solution adaptive document, all business attributes that were archived are passed onto this method for re-initializing the referenced business objects. For more information see “Life-cycle management of adaptive documents” on page 47.

Parameters:

adocId
The adaptive document identifier of the archived adaptive document.

Returns:

Object
The Solution adaptive document remote handle. Must be narrowed before it is used.

Throws:

java.rmi.RemoteException
Wrapped around any thrown exception.

serviceRequest

```
public PBSEventOutput serviceRequest( PBSEventInput event)
                              throws java.rmi.RemoteException
```

This method provides the capability to make a service request on an adaptive document instance. The PBSEventInput, PBSEventOutput classes encapsulate the parameter requirements for the service request.

Returns:

Hashtable
The output, encapsulated in a PBSEventOutput object.

Throws:

java.rmi.RemoteException
The exception description.

serviceRequest

```
public java.util.Hashtable serviceRequest(java.lang.String request,
                                          java.util.Hashtable context,
                                          java.util.Hashtable input,
                                          java.lang.String adocId,
                                          java.lang.String user)
                                   throws java.rmi.RemoteException
```

This brokers the event (request) across the adaptive document instance and all associated activity controllers. First the event is checked to be raised against the adaptive document instance. If the adaptive document is in a state to accept the event, it is consumed and the adaptive document transitions to another state. Next the same event is raised against all associated activities and thus is consumed if it is found to be valid. All these operations happen within a single unit of work.

Parameters:

request
The Event.

context
The context information.

input
The input information.

adocid
The adaptive document identifier.

user
The user invoking the operation. A null can be passed if the user is unknown.

Returns:

Hashtable
The output of the adaptive document and task controller commands.

Throws:

Throws: java.rmi.RemoteException
The exception description.

setSessionContext

```
public void setSessionContext(javax.ejb.SessionContext ctx
                              throws java.rmi.RemoteException
```

SetSessionContext method comment. This method is specified by setSessionContext in the interface javax.ejb.SessionBean.

Parameters:

ctx
javax.ejb.SessionContext

Returns:
void

Throws:

java.rmi.RemoteException
The exception description.

AdocEvents Class

```
java.lang.Object
  |
  +--com.ibm.epic.bfm.ejb.base.AdocEvents
```

public abstract class **AdocEvents**
extends java.lang.Object implements java.io.Serializable

This class is used to transport the adaptive document along with all events that can be raised against it. This class is primarily used to encapsulate adaptive document query calls.

toString

```
public java.lang.String toString()
```

Returns a concatenated string with all the adaptive document attributes.

Overrides:

toString in class java.lang.Object

AdocProxy Class

```
java.lang.Object
  |
  +--com.ibm.epic.bfm.ejb.base.AdocProxy
```

public abstract class **AdocProxy**
extends java.lang.Object

AdocProxy is an abstract base class. Each solution adaptive document must be derived from this class.All the abstract methods in this class has to be implemented by the solution adaptive document classes. The AdocProxy class has a reference to the adaptive document enterprise bean. This reference could be obtained by the child classes using the method getProxy() to use any other services provided by the adaptive document enterprise bean.

archive

```
public java.util.Hashtable archive
                        throws java.rmi.RemoteException
```

Parameters

filterParams
java.util.Hashtable

Returns
Boolean

Throws

java.rmi.RemoteException
The description of the exception.

filter

```
public java.lang.Boolean filter(java.util.Hashtable filterParams
                        throws java.rmi.RemoteException
```

Parameters

filterParams
java.util.Hashtable

Returns
Boolean

Throws

java.rmi.RemoteException
The description of the exception.

getActionList

```
public abstract java.util.Vector getActionList()
                            throws java.rmi.RemoteException
```

This method gets the action list. It must be implemented by the derived class.

Returns
A vection actions list.

Throws

java.rmi.RemotException
A description of the exception.

getAdocId

```
public abstract java.lang.String getAdocId()
                            throws java.rmi.RemoteException
```

This method gets the adaptive document ID.

Returns
A string containing the adaptive document ID.

Throws

java.rmi.RemotException
A description of the exception.

getAdocName

```
public abstract java.lang.String getAdocName()
                  throws java.rmi.RemoteExecption
```

Access method to get the adaptive document name. This method must be implemented by the derived class. A string containing the adaptive document name is returned.

Returns
A string containing the adaptive document ID.

Throws

java.rmi.RemotException
A description of the exception.

getAdocOwner

```
public abstract java.lang.String getAdocOwner()
                  throws java.rmi.RemoteExecption
```

Access method to get the adaptive document owner. This method must be implemented by the derived class. A string containing the adaptive document owner is returned.

Returns
A string containing the adaptive document owner.

Throws

java.rmi.RemotException
A description of the exception.

getAdocState

```
public abstract java.lang.String getAdocState()
                  throws java.rmi.RemoteExecption
```

Access method to get the adaptive document state. This method must be implemented by the derived class.

Returns
A string containing the adaptive document state.

Throws

java.rmi.RemotException
A description of the exception.

getPKString

```
public abstract java.lang.String getPKString()
```

Access method to get the primary key. This method must be implemented by the derived class. The primary key is returned as a string.

getUniqueString

```
public java.lang.String getUniqueString()
                        (java.lang.String type)
                         throws java.rmi.RemoteException
```

Access method to get a unique string for a given type.

Parameters

type
The type for the controller.

Returns

String
The string contains the unique ID.

Throws

java.rmi.RemoteException
The description of the exception.

revive

```
public void revive(java.util.Hashtable adocData
                        throws java.rmi.RemoteException
```

Parameters

filterParams
java.util.Hashtable

Returns
Boolean

Throws

java.rmi.RemoteException
The description of the exception.

setAdocOwner

```
public abstract void setAdocOwner
                       (java.lang.String arg1)
                        throws java.rmi.RemoteException
```

Access method to set the owner of the adaptive document.

Parameters

String
The adaptive document owner.

Returns
Void

Throws

java.rmi.RemoteException
The description of the exception.

setAdocState

```
public abstract void setAdocState
                       (java.lang.String arg1)
                        throws java.rmi.RemoteException
```

Access method to set the type of the adaptive document.

Parameters

String
The adaptive document state.

Returns
Void

Throws

java.rmi.RemoteException
The description of the exception.

unsetInternalState

```
public abstract void unsetInternalState
                        throws java.rmi.RemoteException
```

A clean up method for resetting member variables in the adaptive document. This method must be implemented by the derived class.

Returns
Void

Throws

java.rmi.RemoteException
The description of the exception.

DefaultGenericServiceRequestHandler Class

```
java.lang.Object
  |
  +--com.ibm.epic.timer.DefaultGenericServiceRequestHandler
```

public class **DefaultGenericServiceRequestHandler**
extends DefaultGenericServiceRequestHandler Class

This is the generic service request handler. An entry of DEFAULT when scheduling a service request prompts the TimerDispatcher to instantiate and run one of these. This class uses the timer entries to invole the service request. A subclass of this class can potentially augement the service request parameters. The subclass can be registered as the handler class.

```
public DefaultGenericServiceRequestHandler(java.lang.String namedEvent,
                            java.util.Hashtable context,
                            java.util.Hashtable input,
                            java.lang.String refId,
                            java.lang.String user,
                            com.ibm.epic.bfm.ejb.BFMAdmin bfm)
```

The constructor calls the constructor in the base class and passes the parameters.

java.lang.String namedEvent
The named event.

java.util.Hashtable context
The context

java.util.Hashtable input
The input

java.lang.String refId
The adaptive document ID.

com.ibm.epic.bfm.ejb.BFMAdmin bfm bfm
The reference to the BFMAdmin session bean.

doServiceRequest

```
public void doServiceRequest()
                    throws java.lang.Throwable
```

PBSEventInput Class

```
java.lang.Object
  |
  +--com.ibm.epic.bfm.ejb.base.PBSEventInput
```

public abstract class **PBSEventInput**
extends java.lang.Object implements java.io.Serializable

This class is used to encapsulate the parameters needed to make a service request. An instance of this class needs to be created for every possible event. This is done by sub-classing from this class and providing additional methods to set the input parameters.

PBSEventInput

```
public PBSEventInput(java.lang.String adocType,
                     java.lang.String event,
                     java.lang.String adocId,
                     java.lang.String user)
```

getAdocId

```
public java.lang.String getAdocId()
```

getAdocType

```
public java.lang.String getAdocType()
```

getEvent

```
public java.lang.String getEvent()
```

getEventParams

```
public java.util.Hashtable getEventParams()
```

getUser

```
public java.lang.String getUser()
```

setEventParam

```
protected void setEventParam(java.lang.String paramName,
                             java.lang.Object obj)
                      throws java.lang.NullPointerException
```

Is called from the concrete class to set required input parameters for the service request.

Parameters:

paramName
The key to use for this parameter.

obj
The parameter Object

Throws:

java.lang.NullPointerException
If either of the parameters are null, a NullPointerException is thrown.

PBSEventOutput Class

```
java.lang.Object
  |
  +--com.ibm.epic.bfm.ejb.base.PBSEventOutput
```

public class **PBSEventOutput**
extends java.lang.Object
implements java.io.Serializable

A simple class to encapsulate the return of a service request.

PBSEventOutput

```
public PBSEventOutput(java.lang.String adocType,
                      java.lang.String event,
                      java.lang.String adocId,
                      java.lang.String user,
                      java.util.Hashtable outVals)
```

getAdocId

```
public java.lang.String getAdocId()
```

getAdocType

```
public java.lang.String getAdocType()
```

getEvent

```
public java.lang.String getEvent()
```

getInvokingUser

```
public java.lang.String getInvokingUser()
```

getOutVals

```
public java.util.Hashtable getOutVals()
```

AdocDetails Class

```
java.lang.Object
  |
  +--com.ibm.epic.bfm.ejb.base.AdocDetails
```

public class **AdocDetails**
extends java.lang.Object
implements java.io.Serializable

This is a simple class to transport base adaptive document attributes by Process Broker Services. All adaptive document query methods return the base adaptive document attributes wrapped in this structure.

AdocDetails

```
public AdocDetails(java.lang.String aid,
                   java.lang.String an,
                   java.lang.String as,
                   java.lang.String ao)
```

getAdocId

```
public java.lang.String getAdocId()
```

Access method to get the adaptive document identifier.

getAdocName

```
public java.lang.String getAdocName()
                  throws java.rmi.RemoteExecption
```

Access method to get the adaptive document name. This method must be implemented by the derived class. A string containing the adaptive document name is returned.

getAdocOwner

```
public java.lang.String getAdocOwner()
```

Access method to get the adaptive document owner.

getAdocState

```
public java.lang.String getAdocState()
```

Access method to get the adaptive document state.

getAdocType

```
public java.lang.String getAdocType()
```

Access method to get the adaptive document type.

getPKString

```
public java.lang.String getPKString()
```

Access method to get the primary key. This method must be implemented by the derived class. The primary key is returned as a string.

getUniqueString

```
public java.lang.String getUniqueString()
                        (java.lang.String type)
                         throws java.rmi.RemoteException
```

Access method to get a unique string for a given type.

Parameters

type
The type for the controller.

Returns

String
The string contains the unique ID.

Throws

java.rmi.RemoteException
The description of the exception.

toString

```
public java.lang.String toString()
```

Returns a concatenated string with all the adaptive document attributes.

Overrides:

toString in class java.lang.Object

EventDetails Class

```
java.lang.Object
  |
  +--com.ibm.epic.bfm.ejb.base.EventDetails
```

public class **EventDetails**
extends java.lang.Object
implements java.io.Serializable

This is a simple class to transport an event with its associated activity details.

```
public EventDetails(java.lang.String en,
                    java.lang.String an,
                    java.lang.String ar,
                    java.lang.String au)
```

public java.lang.String _eventName
The event.

public java.lang.String _activityName
The associated Activity Name if an activity event

public java.lang.String _activityRole
The associated Activity Role if an activity event.

public java.lang.String _activityUser
The associated Activity User if an activity event

toString

```
public java.lang.String toString()
```

Returns a concatenated string with all the adaptive document attributes.

Overrides:

toString in class java.lang.Object

TimerServiceBean Class

```
java.lang.Object
  |
  +--com.ibm.epic.bfm.ejb.TimerServiceBean
```

public class **TimerServiceBean**
extends java.lang.Object
implements javaax.ejb.SessionBean

TimerService is a Stateless Session Bean. It is a service to schedule service requests on Process Broker Services. The scheduled service requests are consumed by the TimerDispatcher that invokes listeners or handlers to make

the service request at the scheduled time. The interfaces to schedule requests are available as system commands in the command.xml file. For details refer to "Process Broker Services scheduler" on page 50.

ejbActivate

```
public void ejbActivate()
                    throws java.rmi.RemoteException
```

This method is specified by ejbActivate in the interface javax.ejb.SessionBean.

Returns:
Void

Throws:

java.rmi.RemoteException
The exception description.

ejbCreate

```
public void ejbCreate()
                           throws java.ejb.CreateException
                                  java.rmi.RemoteException
```

This method creates the bean.

Throws:

java.ejb.CreateException
The exception description.

java.rmi.RemoteException
The exception description.

ejbPassivate

```
public void ejbPassivate()
                           throws java.rmi.RemoteException
```

This method is called when the Enterprise bean changes to the passivation state. Cached objects are released. This method is specified by ejbPassivate in the interface javax.ejb.SessionBean.

Throws:

java.rmi.RemoteException
The exception description.

ejbRemove

```
public void ejbRemove()
                           throws java.rmi.RemoteException
```

This method is called when the Enterprise bean is removed. All internal references are cleared and all database connections are closed. This method is specified by ejbRemove in the interface javax.ejb.SessionBean.

Throws:

java.rmi.RemoteException
The exception description.

getAllExpiredTimerEntries

```
public java.util.Vector getAllExpiredTimerEntries()
                            throws java.rmi.RemoteException
```

This method gets all expired service requests. The service request details are returned wrapped in a TEStruct object.

Throws:

java.rmi.RemoteException
The exception description.

getAllProcessableTimerEntries

```
public java.util.Vector getAllProcessableTimerEntries()
                            throws java.rmi.RemoteException
```

This method gets all expired service requests that can be processed. A service request entry can still be processed if the retry count is not equal to zero.

Throws:

java.rmi.RemoteException
The exception description.

getSessionContext

```
public javax.ejb.SessionContext getSessionContext()
                            throws java.rmi.RemoteException
```

Returns
javax.ejb.SessionContext

markUnprocessable

```
public void markUnprocessable(long _timerEntryId)
                            throws java.rmi.RemoteException
```

Mark the timer entry as unprocessable. Sets the number of attempts left to zero.

Parameters:

_timerentryId
The Id for the service request entry.

Throws:

java.rmi.RemoteException
The exception description.

recordAnAttempt

```
public int recordAnAttempt(long _timerEntryId)
                           throws java.rmi.RemoteException
```

Record an attempt to inoke a service request entry. This updates the *lastExecutedTimeStamp* and decrease the retry count.

Parameters:

_timerentryId
The Id for the service request entry.

Throws:

java.rmi.RemoteException
The exception description.

removeTimerEvent

```
public void removeTimerEvent(long Id)
                           throws java.rmi.RemoteException
```

Record an attempt to inoke a service request entry. This updates the *lastExecutedTimeStamp* and decrease the retry count.

Parameters:

Id The Id for the service request entry.

Throws:

java.rmi.RemoteException
The exception description.

scheduleAdocArchival

```
public java.lang.String scheduleAdocArchival
                                    (java.lang.String adocId,
                                     java.lang.String dateTimeStr,
                                     java.lang.String numRetries)
                                  throws java.rmi.RemoteException
```

Schedules an adaptive document for archival. This method is available as a system command in the command.xml. It is recommended to specify a high retry count in the case of transaction rollbacks happening due to high volume of transactions. A retry count of 20 is recommended

Parameters:

java.lang.String adocId
The Id of the adaptive document.

java.lang.String dateTimeStr
The date and time when the document should be archived. [MM:DD:YY] HH:MM AM/PM for example, 4/12/01 1:40 AM, 1:00 PM (1:00 PM today), or null nnnnnn archive after nnnnnn milliseconds of creation. For example, 60000 (fire after 1 min)

java.lang.String numRetries
The number of retry attempts.

Throws:

java.rmi.RemoteException
The exception description.

scheduleAdocInitializationWithGeneratedID

```
public java.lang.String scheduleAdocInitializationWithGeneratedID
                        (java.lang.String adocType,
                        java.lang.String adocOwner
                        java.lang.String namedEvent,
                        java.util.Hashtable input,
                        java.util.Hashtable context,
                        java.lang.String dateTimeStr,
                        java.lang.String numRetries)
                        throws java.rmi.RemoteException
```

This method is used to schedule the creation and initialization of an adaptive document. This method is available as a system command in the command.xml file. It can be used to create an adaptive document from within the transition of another document. After the document is created, a service request is made on the document with the event passed, using the input and context hashtables. This event can potentially be used to initialize the created adaptive document by creating BO's, setting the BO references in the adaptive document and so on.

Parameters:

java.lang.String adocType
The type of adaptive document to be created.

java.lang.String adocOwner
The user creating the adaptive document.

java.lang.String namedEvent
The event to be raised on the adaptive document. If null the default is the document is initialized.

java.util.Hashtable input
The input Hashtable to be used for the event.

java.util.Hashtable context
The context Hastable to be used for the event.

java.lang.String dateTimeStr
The date and time when the document should be archived. [MM:DD:YY] HH:MM AM/PM for example, 4/12/01 1:40 AM, 1:00 PM (1:00 PM today), or null nnnnnn archive after nnnnnn milliseconds of creation. For example, 60000 (fire after 1 min)

java.lang.String numRetries
The number of retry attempts.

Throws:

java.rmi.RemoteException
The exception description.

scheduleAdocInitializationWithGivenID

```
public java.lang.String scheduleAdocInitializationWithGivenID
                        (java.lang.String adocType,
                        java.lang.String adocId,
                        java.lang.String adocOwner
                        java.lang.String namedEvent,
                        java.util.Hashtable input,
                        java.util.Hashtable context,
                        java.lang.String dateTimeStr,
                        java.lang.String numRetries)
                        throws java.rmi.RemoteException
```

This method is used to schedule the creation and initialization of an adaptive document. This method is available as a system command in the command.xml file. It can be used to create an adaptive document from within the transition of another document. After the document is created, a service request is made on the document with the event passed, using the input and context hashtables. This event can potentially be used to initialize the created adaptive document by creating BO's, setting the BO references in the adaptive document and so on.

Parameters:

java.lang.String adocType
The type of adaptive document to be created.

java.lang.string adocId
The adaptive document ID to used as the key for the created adaptive document or null. If null, the Business Flow Manager generates the ID.

java.lang.String adocOwner
The user creating the adaptive document.

java.lang.String namedEvent
The event to be raised on the adaptive document. If null the default is the document is initialized.

java.util.Hashtable input
The input Hashtable to be used for the event.

java.util.Hashtable context
The context Hastable to be used for the event.

java.lang.String dateTimeStr
The date and time when the document should be archived. [MM:DD:YY] HH:MM AM/PM for example, 4/12/01 1:40 AM, 1:00 PM (1:00 PM today), or null nnnnnn archive after nnnnnn milliseconds of creation. For example, 60000 (fire after 1 min)

java.lang.String numRetries
The number of retry attempts.

Throws:

java.rmi.RemoteException
The exception description.

scheduleAdocRemoval

```
public java.lang.String scheduleAdocRemoval
                        (java.lang.String adocId,
                        java.lang.String dateTimeStr,
                        java.lang.String numRetries)
                        throws java.rmi.RemoteException
```

Schedules an Adoc to be removed. This method is available as a system command in the command.xml file. It is recommended to assign a high retry count in the case of transaction rollbacks happening due to a high volume of transactions.. A retry count of 20 is suggested.

Parameters:

java.lang.string adocId
The ID of the adaptive document to be removed.

java.lang.String dateTimeStr
The date and time when the document should be archived. [MM:DD:YY] HH:MM AM/PM for example, 4/12/01 1:40 AM, 1:00 PM (1:00 PM today), or null nnnnnn archive after nnnnnn milliseconds of creation. For example, 60000 (fire after 1 min)

java.lang.String numRetries
The number of retry attempts.

Throws:

java.rmi.RemoteException
The exception description.

scheduleServiceRequestWithDefaultHandler

```
public java.lang.String scheduleServiceRequestWithDefaultHandler
                        (java.lang.String bfmRefId,
                         java.lang.String bfmRefType,
                         java.lang.String namedEvent,
                         java.util.Hashtable input,
                         java.util.Hashtable context,
                         java.lang.String dateTimeStr,
                         java.lang.String numRetries)
                       throws java.rmi.RemoteException
```

This method is used to schedule a service request. This method is available as a system command in the command.xml file.

Parameters:

java.lang.string bfmRefId
The adaptive document or Activity ID.

java.lang.String bfmRefType
The bfmRefType of the adaptive document or activity.

java.lang.String namedEvent
The event to be raised on the adaptive document. If null the default is the document is initialized.

java.util.Hashtable input
The input Hashtable to be used for the event.

java.util.Hashtable context
The context Hastable to be used for the event.

java.lang.String dateTimeStr
The date and time when the document should be archived. [MM:DD:YY] HH:MM AM/PM for example, 4/12/01 1:40 AM, 1:00 PM (1:00 PM today), or null nnnnnn archive after nnnnnn milliseconds of creation. For example, 60000 (fire after 1 min)

java.lang.String numRetries
The number of retry attempts.

Throws:

java.rmi.RemoteException
The exception description.

scheduleServiceRequestWithGivenHandler

```
public java.lang.String scheduleServiceRequestWithGivenHandler
                        (java.lang.String bfmRefId,
                         java.lang.String bfmRefType,
                         java.lang.String namedEvent,
                         java.util.Hashtable input,
                         java.util.Hashtable context,
```

```
                                java.lang.String handlerClass
                                java.lang.String dateTimeStr,
                                java.lang.String numRetries)
                              throws java.rmi.RemoteException
```

This method is used to schedule a service request. This method is available as a system command in the command.xml file.

Parameters:

java.lang.string bfmRefId
The adaptive document or Activity ID.

java.lang.String bfmRefType
The bfmRefType of the adaptive document or activity.

java.lang.String namedEvent
The event to be raised on the adaptive document. If null the default is the document is initialized.

java.util.Hashtable input
The input Hashtable to be used for the event.

java.util.Hashtable context
The context Hastable to be used for the event.

java.lang.String handlerClass
The fully qualified handler or listener class, or Default to use the default handlers.

java.lang.String dateTimeStr
The date and time when the document should be archived. [MM:DD:YY] HH:MM AM/PM for example, 4/12/01 1:40 AM, 1:00 PM (1:00 PM today), or null nnnnnn archive after nnnnnn milliseconds of creation. For example, 60000 (fire after 1 min)

java.lang.String numRetries
The number of retry attempts.

Throws:

java.rmi.RemoteException
The exception description.

scheduleServiceRequestWithGivenHandlerbySpecifiedUser

```
public java.lang.String scheduleServiceRequestWithGivenHandlerbySpecifiedUser
                       (java.lang.String bfmRefId,
                        java.lang.String bfmRefType,
                        java.lang.String namedEvent,
                        java.util.Hashtable input,
                        java.util.Hashtable context,
                        java.lang.String handlerClass
                        java.lang.String dateTimeStr,
```

```
                                java.lang.String numRetries)
                                java.lang.String user
                              throws java.rmi.RemoteException
```

This method is used to schedule a service request. This method is available as a system command in the command.xml file.

Parameters:

java.lang.string bfmRefId
The adaptive document or Activity ID.

java.lang.String bfmRefType
The bfmRefType of the adaptive document or activity.

java.lang.String namedEvent
The event to be raised on the adaptive document. If null the default is the document is initialized.

java.util.Hashtable input
The input Hashtable to be used for the event.

java.util.Hashtable context
The context Hastable to be used for the event.

java.lang.String handlerClass
The fully qualified handler or listener class, or Default to use the default handlers.

java.lang.String dateTimeStr
The date and time when the document should be archived. [MM:DD:YY] HH:MM AM/PM for example, 4/12/01 1:40 AM, 1:00 PM (1:00 PM today), or null nnnnnn archive after nnnnnn milliseconds of creation. For example, 60000 (fire after 1 min)

java.lang.String numRetries
The number of retry attempts.

java.lang.String user
The user that needs to launch this request or null if the user is not known.

Throws:

java.rmi.RemoteException
The exception description.

setSessionContext

```
public void setSessionContext(javax.ejb.SessionContext ctx
                              throws java.rmi.RemoteException
```

This method is specified by setSessionContext in the interface javax.ejb.SessionBean.

Parameters:

ctx
javax.ejb.SessionContext

Returns:
void

Throws:

java.rmi.RemoteException
The exception description.

UserCondition Interface

```
com.ibm.epic.bfm.controller
```

public interface **UserCondition**

evaluate

```
public java.lang.Boolean evaluate(java.lang.String event,
                                  java.util.Hashtable context,
                                  java.util.Hashtable input
```

Parameters:

java.lang.String event
The event to be raised on the adaptive document. If null the default is the document is initialized.

java.util.Hashtable input
The input Hashtable to be used for the event. The adaptive document reference can be passed in this Hashtable if the adaptive document attributes are required in order to write the logic.

java.util.Hashtable context
The context Hastable to be used for the event.

Returns:

Boolean
Returns True or False.

Notices

This information was developed for products and services offered in the U.S.A.

IBM may not offer the products, services, or features discussed in this document 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 document. The furnishing of this document 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 publication. IBM may make improvements and/or changes in the product(s) and/or program(s) described in this publication 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 Limited
Intellectual Property Department
Hursley Park
Winchester SO21 2JN
United Kingdom

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 Program 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 measures may have been made on development-level systems, and there is no guarantee that these measurements will be the same on generally available system. Furthermore, some measurements may have been estimated through extrapolation. Actual results may vary. Users of this document should verify the application data of 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 claim related to non-IBM products. Questions on capabilities of non-IBM products should be addressed to the suppliers of those products.

This information contains examples of data and reports used in daily business operations. To illustrate them as completely as possible, the examples include names of individuals, companies, brands, and products. All of these names are fictitious and any similarity to the names and addresses used by an actual business enterprise is entirely coincidental.

Glossary of Terms and Abbreviations

This glossary defines terms and abbreviations used in Business Integrator. If you do not find the term you are looking for, see the *IBM Dictionary of Computing*, New York: McGraw-Hill, 1994, or refer to the Web Site at http://www.ibm.com/ibm/terminology, which consolidates several of the main glossaries created for IBM products in one convenient location, including:

- Glossary of Computing Terms
- DB2 Glossary
- Tivoli Glossary

The following cross-references are used in this glossary:

Contrast with This refers you to a term that has an opposed or substantively different meaning.

See This refers you to (a) a related term, (b) a term that is the expanded form of an abbreviation or acronym, or (c) a synonym or more preferred term.

A

Active Directory Service Interfacer (ADSI). A means for client applications to use a common set of interfaces to communicate with and control any server that implements them. This allows a single client application to configure a number of different servers because it is shielded from API details specific to each server.

adapter. An element of a solution that provides semantic adaptations based on specific interchange schema specified by the message set and that allows legacy applications to communicate with the Business Integrator system.

During run time, the adapter with MQSeries Adapter Kernel and MQSeries act as the *Information Delivery Manager*.

The adapter is output from MQSeries Adapter Builder under Solution Studio. See the MQSeries Adapter Builder documentation for a full definition of adapter.

Contrast with *Business Process Integration Adapter*.

adaptive document. Part of how *Process Choreography* is implemented on Business Flow Manager. The adaptive document includes state machine functionality and persistent data. It can be realized through several approaches.

Different views of the business content are rendered to the end user by the *Interaction Manager*.

application adapter. See *adapter*.

Application Service Provider (ASP). A company that offers subscription services for applications and related services on a pay-per-use basis. ASPs host, manage and maintain applications at their own site and make them available via the Web. This enables smaller companies or those with limited budgets to take full advantage of the latest information technology.

artifacts. The name for the set of all deployable files that make up a solution. Artifacts can be:

- Executables, for example, enterprise beans, Java beans, Java Server Pages, servlets, MQSeries Adapter Offering adapters, and various class files
- Configuration files, for example, MQSeries configuration files that control queues, LDIF (LDAP configuration) files, MQSeries Integrator MRP files, Policy Director map files, and files that control personalization

- Other content, for example, HTML files

You create artifacts by means of Solution Studio and its associated tools such as VisualAge for Java, MQSeries, MQSeries Integrator, MQSeries Adapter Offering, WebSphere Application Server, Partner Agreement Manager, DataInterchange and others. See *solution package*.

ASP. See *Application Service Provider*

Audit Log. The log of messages that are sent between applications and adapters.

Audit Log server. The Business Integrator component that allows the storage and retrieval of audit log information in a DB2 database.

B

B2B. See *business-to-business*.

B2C. See *business-to-customer*.

base machine. The machine that contains the *Topology Repository*, and the Trust and Access Manager facility. Depending on the topology selected, the base machine may also contain other facilities.

BO. See *business object*.

BOD. See *Business Object Document*.

broker. See *message broker*.

business flow. The business processes, at the task level, that drive the business.

Business Flow Manager. The Business Integrator component that controls the flow of business processes. It can be invoked programmatically by Interaction Manager, or via a message from a gateway or an Endpoint. It can invoke Endpoints.

Business Flow Manager provides a platform for:

- *Microflows*
- Data objects
- Workflow enterprise beans that invoke and communicate with MQSeries Workflow
- *JMS Listener*
- *Worker beans*

Business Integrator. The short name for IBM WebSphere Business Integrator.

Business Integrator Log Client. Software that is installed with most of the facilities of Business Integrator, and which allows logging at those facilities.

business object. Data and application objects that persist data for business entities or tasks that are used to populate messages exchanged with other components during run time. Business objects are reusable Enterprise Java Beans (EJB) that typically implement a business entity or task. A business object has both data and function. For example, a request for quotation (RFQ) is a business object that might have a unique identifier, a vendor, and one or more parts.

Business objects are also known as business data objects.

Business Object Document (BOD). A representation of a standard business process that flows within an organization or between organizations. Examples are: add purchase order, show product availability, and add sales order. BODs are defined by the *Open Applications Group* using XML.

business process. The step-by-step flow of information and actions within one organization or between two or more trading partners. A business process can be as simple or complex as needed to meet the business needs of the organization or trading partners. See *task.* There are several types of business processes. See *private process* and *public process*.

Business Process Integration Adapter. Software that enables Partner Agreement Manager to work with the rest of Business Integrator. It packages XML message payloads from a public process into message formats that can be transported by Information Delivery Manager to Endpoints or to the Business Flow Manager. It packages messages from Endpoints or the Business Flow Manager into message formats that can be sent

through a public process. The Business Process Integration Adapter manages the correlation information that is necessary to bind public processes in Partner Agreement Manager with business processes in the Business Flow Manager.

Business Process Managers. The Business Integrator components that control Business Integrator. The Business Process Managers are *Trust and Access Manager, Business Flow Manager, Interaction Manager, Information Delivery Manager, and Solution Manager*.

business-to-business (B2B). Electronic commerce where a buyer organization buys goods or services from a merchant organization.

business-to-customer (B2C). Electronic commerce where a consumer buys goods or services from a merchant.

C

capacity unit. A measure of the number of Symmetrical Multiprocessors (SMP) in a machine. This is used to calculate the license requirements when you purchase Business Integrator.

Certificate Authority (CA). 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 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.

channel. In Partner Agreement Manager, an encapsulation of all the processing information needed to send messages to a trading partner's system, and to translate data from a trading partner into Partner Agreement Manager messages. All Partner Agreement Manager installations have the *PAM-to-PAM channel* installed. Available non-Partner Agreement Manager channels include the *RosettaNet* and *XML channel*.

ClearCase repository. In Solution Studio, the storage place for the *solution artifacts*. Rational ClearCase manages version control for all of the artifact files created for each solution.

common systems administration (CSA). Tools and technology used to implement the Product Console Launchpad and solution management framework in Business Integrator. The use of CSA provides a consistent look and feel within the system management consoles of Business Integrator and other IBM products.

correlation identifier. A field in a message that provides a means of identifying related messages. Correlation identifiers are used, for example, to match request messages with their corresponding reply message.

CRM. See *Customer Relationship Management*.

CSA. See *common systems administration*.

Customer Relationship Management (CRM). The systems and infrastructure required to analyze, capture, and share all parts of the customer's relationship with the enterprise. From a strategy perspective, CRM represents a process for measuring and allocating organizational resources to those activities that have the greatest return and impact on profitable customer relationships.

D

DataInterchange. A component of the Business Integrator that performs messaging between the trusted zone and an EDI network or virtual private network. DataInterchange reformats data for transmission via one or more channels, transforms data, supports registration of trading partners, performs auditing and reporting, supports extensive customization, and provides APIs for administrative, logging and command procedures.

DataInterchange can accept messages from the untrusted zone, perform appropriate processing and send the message to Endpoints or to the Business Flow Manager.

Endpoints or the Business Flow Manager can send messages to DataInterchange which cause DataInterchange to execute one or more command procedures to perform one or more actions, for example, to log on to a mailbox, and then to check for messages.

DataInterchange adapter. Software that enables DataInterchange to work with the rest of Business Integrator.

data object. In the Business Integrator programming model, a special type of command that encapsulates access to a data store.

Data Universal Numbering System (DUNS). A system in which internationally recognized nine-digit numbers are assigned and maintained by Dun & Bradstreet to uniquely identify worldwide businesses.

DB2. An IBM relational database management system that is available as a licensed program. Programmers and users of DB2 can create, access, modify, and delete data in relational tables using a variety of interfaces.

DB2 XML Extender. An extension to DB2 that provides data types that let you store XML documents in DB2 databases and functions that assist you in working with these structured documents. Entire XML documents can be stored in DB2 databases as character data or stored as external files but still managed by DB2. Retrieval functions allow you to retrieve either the entire XML document or individual elements or attributes.

DCE. See *distributed computing environment.*

Demilitarized Zone (DMZ). In network security, a network that is isolated from, and serves as a neutral zone between, a trusted network (for example, a private intranet) and an untrusted network (for example, the Internet). One or more secure gateways usually control access to the DMZ from the trusted or the untrusted network.

deployment. The process of making all the elements of a solution available to the run-time system. Compare with *publishing*.

deployment application. Part of the run time that unzips the solution package into the constituent artifacts and that moves the artifacts to the correct location on each machine in the run time environment according to the topology. The deployment application is invoked by the *Solution Deployment Wizard*. See *artifact* and *solution package*.

Digital Certificate. A form of electronic ID. The digital certificate facilitates unique identification of the entity that holds it. It is issued by a Certificate Authority (CA).

distributed computing environment (DCE). A product that assists in networking by providing such functions as authentication, directory service (DS), and remote procedure call (RPC).

DMZ. See *Demilitarized Zone*.

Document Type Definition (DTD). A file associated with XML documents that defines how the markup tags should be interpreted by the application using the document.

DUNS. See *Data Universal Numbering System*.

Dynamic MBean. An MBean that implements the DynamicMBean interface, so called because certain elements of its instrumentation can be controlled at runtime. See *Managed Bean* and contrast with *Standard MBean.*

E

EAI. See *enterprise application integration*.

e-business process integration (e-BPI). A system that enables companies to create, execute, and manage processes that span diverse applications, enterprises, and people, and to manage those processes—as well as the components that support those processes—as a unified, extensive, flexible solution.

Business Integrator is an e-business process integration system.

EDI. See *Electronic Data Interchange*.

EJB. See *Enterprise Java Bean*.

Electronic Data Interchange (EDI). A method of transmitting business information over a network, between trading partners who agree to follow approved national or industry standards in translating and exchanging information.

e-market. Where business-to-business buyers and sellers meet to trade in a virtual market.

Endpoint. A machine within the trusted zone that contains one or more *Endpoint applications*, each with an *application adapter* that allows communication with the Business Integrator system. A Business Integrator system can contain one or more Endpoint machines. Each Endpoint machine contains a single *Endpoint facility*.

Endpoint application. A business application that resides on an Endpoint machine together with an *application adapter*. Endpoint applications are typically called legacy applications or enterprise applications. Examples of such applications are SAP applications.

Endpoint facility. A significant portion of Business Integrator's run time functionality that is installed on each Endpoint machine. An Endpoint facility provides access via one or more adapters to Endpoint applications, and enable messaging within Business Integrator. The Endpoint facility also helps support deployment and management of the solution.

Endpoint machine. Synonymous with *Endpoint*.

element. See *solution element*.

enterprise application integration (EAI). The integration of disparate systems and applications across an enterprise, and the interoperability of complementary systems and applications between enterprises.

Enterprise configuration. The version of Business Integrator used by large enterprises. Enterprise configuration provides capabilities for more complex interactions with trading partners including EDI capability and generalized access to a Web Application server. Compare with *Entry configuration.*

Enterprise Java Bean (EJB). A Java API that defines a component architecture for multi-tier client/server systems. EJB systems allow developers to concentrate on the business architecture of a model, instead of programming the connections between components. EJB systems are platform-independent and object-oriented, and can be implemented into existing systems with minimal recompiling and configuring.

entity bean. A reusable Java component that is built using the Java Beans technology. Entity beans model business concepts that can be expressed as nouns. Entity beans represent data, so a change to an entity bean results in a change on a database. Entity beans are persistent; if the container in which an entity bean is hosted crashes, the entity bean and any remote references survive the crash. Contrast with *session bean*.

Entry configuration. The version of Business Integrator used by small companies and departments of large enterprises. Entry configuration enables trading partners to participate in e-markets with a limited initial investment and relatively low level of complexity. It allows peer-to-peer or spoke-to-hub participation in electronic markets. Compare with *Enterprise configuration*.

EPAC. See *Extended Privilege Attribute Certificate*.

event. In the context of Partner Agreement Manager, 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 and which triggers a *public process*.

Exception Management server. The Business Integrator component that allows the storage and retrieval of exception information in a DB2 database.

Extended Privilege Attribute Certificate (EPAC). A certificate that contains authorization information specific to the user, for example, details of groups to which the user belongs. EPACs are used to authorize users; that is, to

help a server decide whether users should be granted access to resources that the server manages.

Extensible Markup Language (XML). A standard metalanguage for defining markup languages that was derived from and is a subset of Standard Generalized Markup Language (SGML). XML omits the more complex and less-used parts of SGML and makes it much easier to (a) write applications to handle document types, (b) author and manage structured information, and (c) transmit and share structured information across diverse computing systems. The use of XML does not require the robust applications and processing that is necessary for SGML.

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 get information from an enterprise system or listen for an event in the enterprise system. See also *adapter, private process*.

F

facility. In Business Integrator, an indivisible unit of installation that must be completely installed on one machine. A machine contains one or more facilities. Several facilities installed on a single machine may contain a number of common components, in which case those base components are shared amongst the facilities and not multiply installed.

FDL. See *MQSeries Workflow Definition Language.*

firewall. A functional unit that protects and controls the connection of one network to other networks. The firewall (a) prevents unwanted or unauthorized communication traffic from entering the protected network and (b) allows only selected communication traffic to leave the protected network.

G

gateway. A type of component of the Business Integrator that performs messaging between the trusted zone and the Internet or an EDI network.

There are two gateways:

- Partner Agreement Manager, for the Internet
- DataInterchange, for EDI

Global Secure Toolkit (GSK). A toolkit for managing digital certificates used in implementing Secure Sockets Layer (SSL) security.

H

HTTP. See *Hypertext Transfer Protocol*.

HTTP Server. The component of WebSphere Application Server that provides secure Web Server functionality.

Hypertext Transfer Protocol (HTTP). The protocol used by the Internet to transfer HTML and other information from servers to browsers and other servers.

I

Information Delivery Manager. The Business Integrator component that sends messages:

- Between the gateways and the Business Flow Manager
- Between the Business Flow Manager and Endpoints
- Between the gateways and Endpoints

The messages are typically in a common canonical format, that is, in an application-neutral format such as a *Business Object Document* in XML.

The Information Delivery Manager can perform assured delivery of messages, transformation of data elements and related functionalities, routing of messages, and message brokering. This functionality is performed by MQSeries Adapter Kernel with adapters that you build with

MQSeries Adapter Builder, along with MQSeries for assured delivery, and optionally MQSeries Integrator for message brokering services such as complex routing, data transformation, and data mediation.

instrument. In application or system software, to use monitoring functions to provide performance and other information to a management system.

instrumentation. In application or system software, either (a) monitoring functions that provide performance and other information to a management system or (b) the use of monitoring functions to provide performance and other information to a management system.

Interaction Manager. The Business Integrator component that helps to render a view of a business entity that is appropriate to the role of the end user based on their authorization and the point in the business process. Interaction Manager gets the content of the view that it will render from the Business Flow Manager, as the result of the Business Flow Manager's processing.

How the content is rendered depends on the target device used to view the content. For example, presentation on a Web browser might be different from the presentation on a personal digital assistant.

Interaction Manager is not involved when the content of the view is delivered by one of the gateways.

Internet Inter-ORB Protocol (IIOP). A protocol used for communication between CORBA object request brokers.

J

J2EE Connector Architecture. An architecture for the integration of J2EE products with enterprise information systems. The architecture has two parts: a resource adapter provided by an enterprise information system vendor, and the J2EE product that allows this resource adapter to plug in.

Java 2 Platform, Enterprise Edition (J2EE platform). An environment for developing and deploying enterprise applications. The J2EE platform consists of a set of services, APIs, and protocols that provide functionality for developing multi-tiered, Web-based applications.

Java Authentication and Authorization Service (JAAS). A Java package that enables services to authenticate users and enforce access controls upon them.

Java Cryptography Extension (JCE). A framework and implementations for encryption, key generation and key agreement, and Message Authentication Code (MAC) algorithms. JCE is used by Partner Agreement Manager for certificate-based authentication

Java Database Connectivity (JDBC). An industry standard for database-independent connectivity between the Java platform and a wide range of databases. JDBC provides a call-level API for SQL-based database access.

Java Development Kit (JDK). A software package used to write, compile, debug, and run Java applets and applications.

Java Management Extensions (JMX). A means of doing management of and through Java technology. JMX was developed through the Java Community ProcessSM program, by Sun Microsystems, Inc. and some leading companies in the management field. JMX is a universal, open extension of the Java programming language for management that can be deployed across all industries, wherever management is needed.

Java Message Service (JMS). An API for using enterprise messaging systems such as IBM MQSeries.

Java Naming and Directory Interface (JNDI). A set of application programming interfaces that assist with interfacing to multiple naming and directory services.

Java Runtime Environment (JRE). A subset of the Java Development Kit (JDK) comprising the Java Virtual Machine (JVM), the Java core classes, and supporting files.

Java Secure Socket Extension (JSSE). A Java package that enables secure Internet communications. It implements a Java version of the Secure Sockets Layer (SSL) and Transport Layer Security (TSL) protocols and supports data encryption, server authentication, message integrity, and optionally client authentication. JSSE is used by Partner Agreement Manager for certificate-based authentication

Java Server Pages (JSP). An extensible Web technology that uses template data, custom elements, scripting languages, and server-side Java objects to return dynamic content to a client. The template data is typically HTML or XML elements, and the client is often a Web browser.

Java Virtual Machine (JVM). A software implementation of a central processing unit (CPU) that runs compiled Java code (applets and applications).

JCE. See *Java Cryptography Extension*

JCX. A deprecated abbreviation for *J2EE Connector Architecture*.

JDBC. See *Java Database Connectivity*.

JDK. See *Java Development Kit*.

JMS. See *Java Message Service*.

JMS Listener. Part of the Business Flow Manager that is started as a WebSphere service. The JMS Listener, in concert with a *worker bean*, decides which enterprise bean in the Business Flow Manager to invoke. The JMS Listener monitors a JMS queue that is associated with a worker bean. It receives the message from the queue and passes it to the worker bean, which then determines the enterprise bean to invoke based on the content of the message.

The JMS Listener is configured through its own XML file, and in WebSphere's Class of Service Naming service.

JMX. See *Java Management Extensions*.

JNDI. See *Java Naming and Directory Interface*.

JRE. See *Java Runtime Environment*.

JSP. See *Java Server Pages*.

JSSE. See *Java Secure Socket Extension*.

JVM. See *Java Virtual Machine*.

L

LDAP. See *Lightweight Directory Access Protocol*.

Lightweight Directory Access Protocol (LDAP). An open protocol that (a) uses TCP/IP to provide access to directories that support an X.500 model and (b) does not incur the resource requirements of the more complex X.500 Directory Access Protocol (DAP). Applications that use LDAP (known as directory-enabled applications) can use the directory as a common data store for retrieving information about people or services, such as e-mail addresses, public keys, or service-specific configuration parameters.

Lightweight Third Party Authentication (LTPA). An authentication framework that allows single sign-on across a set of Web servers that fall within an Internet domain.

logical topology view. A view of a topology that shows a tree structure in terms of the facilities of Business Integrator and their base products. See *topology* and *facility*.

M

Managed Bean (MBean). According to the Java Management Extensions (JMX) specification, the Java objects that implement resources and their *instrumentation* are called Managed Beans, or MBeans for short. MBeans must follow the design patterns and interfaces defined in the instrumentation level of the JMX specification. This ensures that all MBeans provide the instrumentation of managed resources in a standardized way. MBeans are manageable by

any JMX agent, but they may also be managed by non-compliant agents that support the MBean concept. See also *Java Management Extensions*, *Standard MBean*, and *Dynamic MBean*.

MBean Server. A set of services for handling *MBeans*.

message broker. (1) A set of executing processes hosting one or more message flows. MQSeries Integrator is an example of a message broker. (2) The Business Integrator facility that works with existing messaging transports by adding both routing intelligence and the ability to convert data from one protocol to another. It analyzes a message to determine the application to receive it (rules engine). It then converts the data into the structure that the receiving application requires.

microflow. Part of a *solution*, that is modeled as a Java Service Adapter in MQSeries Adapter Builder. A microflow is deployed as stateless session beans, Java beans, and Java classes, which at run time are part of the Business Flow Manager, running under WebSphere Application Server. A microflow can also can invoke Endpoints, MQSeries Integrator and MQSeries Workflow.

Microsoft Management Console (MMC). An extensible user interface that provides an environment for running management applications structured as components called snap-ins.

middleware. The software that provides the links between applications.

MMC. See *Microsoft Management Console*.

MQSeries. An IBM licensed program that provides reliable message queuing and associated services across a range of platforms.

MQSeries Adapter Builder (MQAB). One of the MQAO set of products, MQAB uses a visual interface to help build an adapter for virtually any application and to build *microflows*.

MQSeries Adapter Kernel (MQAK). One of the MQSeries Adapter Offering set of products that provides common runtime services. In the Business Integrator run time, MQAK, together with MQSeries, and optionally MQSeries Integrator, acts as the Information Delivery Manager.

MQSeries Adapter Offering (MQAO). A set of application integration products that work with MQSeries messaging to reduce the risk, complexity and cost of managing point-to-point application integration. MQAO allows you to create adapters that use a standard interface that remains stable even though the application changes. The interface is typically based on Business Object Documents (BOD), message format standards defined by the Open Applications Group Inc (OAG).

The MQAO is a set of products that includes MQSeries Adapter Builder (MQAB) and MQSeries Adapter Kernel (MQAK).

MQSeries classes for Java Message Service (JMS) (MQ JMS). A set of Java classes, that implement Sun's Java Message Service (JMS) interfaces to enable JMS programs to access MQSeries systems.

MQSeries Integrator (MQSI). A product that works with MQSeries messaging, extending its basic connectivity and transport capabilities to provide a powerful message broker solution driven by business rules.

MQSeries Integrator User Name Server. A component of MQSI that can be used to provide authentication of users and groups performing publish/subscribe operations. At least one of these may be used for each domain, to manage the access paths to resources.

MQSeries Publish/Subscribe. An MQSeries product that allows you to decouple the provider of information from the consumers of the information. An application can send information to a destination managed by MQSeries Publish/Subscribe, which deals with the distribution of the information.

MQSeries Workflow. An MQSeries product that manages *workflow*. MQSeries Workflow is used to design, refine, document, and control business processes.

MQSeries Workflow Definition Language (FDL). The language used to exchange MQSeries Workflow information between MQSeries Workflow systems.

N

node. In Business Integrator one of the points in a *topology view*. Depending on the type of view, a node might correspond to a machine, *facility*, base product, *solution element*, or *solution artifact*.

non-repudiation. In business-to-business communication the ability of the recipient to prove who sent a message based on the contents of the message. This can derive from the use of a digital signature on the message, which links the sender to the message.

O

Object Management Group (OMG). A group of vendors formed for the purpose of creating a standard architecture for distributed objects in networks. The architecture that resulted is the Common Object Request Broker Architecture (Common Object Request Broker Architecture). See also *XML Metadata Interchange*.

Open Applications Group (OAG). A non-profit industry consortium comprised of many prominent stakeholders in the business software component interoperability arena. The OAG defines *Business Object Documents (BOD)*.

P

PAM Proxy Server. Software that resides in the DMZ and which performs access control on inbound messages from business partners to Business Integrator and on outbound messages from Business Integrator to business partners.

PAM-to-PAM channel. A *channel* that is ready-installed in Partner Agreement Manager.

Partner Agreement Connect. A limited licence version of Partner Agreement Manager that allows a customer to participate in a public process defined by a trading partner, but does not allow the authoring of new public processes.

Partner Agreement Manager.

The Business Integrator component that implements the *public process* and *trading partner agreements*.

Partner Agreement Manager is a separately purchased, optional component, of both the Entry configuration and Enterprise configuration of Business Integrator.

Partner Agreement View. (1) A product that provides an interface that allows the seamless integration of Partner Agreement Manager with Web applications. (2) The Business Integrator facility that encapsulates Partner Agreement View.

physical topology view. A view of a topology that shows a tree structure in terms of the machines in the topology, the facilities installed on those machines, and their base products. See *topology* and *facility*.

Platform Console. The Business Integrator console used to monitor and manage the Business Process Managers and components of Business Integrator and, through the Solution Deployment Wizard to deploy solution packages onto the runtime system.

point-to-point messaging. Data transmission between two locations without the use of any intermediate display station or computer.

predefined topology. A topology, corresponding to a tested configuration, that can be selected at installation time. A number of predefined topologies are shipped with Business Integrator to allow for different business needs and complexity of solution.

private process. A trading partner's internal sequence of actions for its steps in the *public process*. Although all trading partners in a public process see and agree to its flow, the trading partner that develops a private process is the only one that can ever see it.

process choreography capability. The name of the capability in *Business Integrator* that:

- Aggregates business content that is managed by a variety of underlying processes (also known as *tasks*). One underlying process might be performed by *MQSeries Workflow*, another might be performed by an *Endpoint application*. Processes can be performed by different organizations.
- Provides the output of an underlying process to one or more other underlying processes that might need it as an input.
- Based on the states of the underlying processes, assigns one state to the overall business process.
- Assigns security privileges to each of the participants across the set of processes.
- Dynamically delivers different views of the business content, based on the point in the business process at which the business content is accessed and on the role of the participant who accesses it.
- Manages the life cycle of the business process.

In summary, the process choreography capability coordinates the set of underlying processes that make up a business process, to align the business process with changing business conditions. Many things can happen at any time that can affect what the business process should do next. To align the business process with changing business conditions, the process choreography capability synchronizes information across multiple underlying processes. It maintains the state of the business process apart from the main process.

The process choreography capability is key to understanding how Business Integrator adds value *beyond* the value of the individual underlying products and technologies that are part of Business Integrator.

The *adaptive document* is part of how the process choreography capability is implemented.

process state. The current state of a business process.

Product Console Launchpad. The graphical user interface used to monitor the runtime system of Business Integrator and launch the management consoles of base products of Business Integrator. The Product Console Launchpad provides a wizard to facilitate the addition of new product consoles to the launchpad.

program client. A Business Integrator client that provides for the automated interchange of business documents with trading partners by responding to program requests and maintaining a relationship with requesting programs.

project. In the build time of Business Integrator, the project organizes and contains all *artifacts* of a single solution. The project and its artifacts are stored in the Clear Case server and in the WebSphere Studio Project, on a mapped network drive that can be shared by a team creating a solution together. You employ the project to organize the work-in-progress files during creation of a solution and the artifacts that result when creation is complete. Project is a WebSphere Studio term.

public process. The step-by-step flow of information and actions between two or more trading partners. One trading partner develops the public process, and all trading partners involved review and accept the process before it is implemented. The trading partner that designs a process is its owner. See *private process*.

publishing. The process of preparing a *solution package* using Solution Studio. The solution package is deployed to the runtime system by the *deployment application*. See also *deployment*.

Publishing Wizard. The wizard within Solution Studio that is used to prepare a *solution package*. Contrast with *Solution Deployment Wizard*.

publish/subscribe. See *MQSeries Publish/Subscribe*.

Q

QoS. Quality of service.

queue manager. (1) A program that provides queuing services to applications. It provides an application programming interface to enable programs to access messages on the queues that the queue manager owns. (2) An object that defines the attributes of a particular queue manager.

R

Rational ClearCase. A product used by Solution Studio to enforce version control, provide change management, and as a repository for solution templates, elements, and artifacts. ClearCase is jointly developed between IBM and Rational.

receiver channel. A channel that moves messages from the target to the source machine.

Remote Method Invocation (RMI). A distributed object model for Java program to Java program, in which the methods of remote objects written in the Java programming language can be invoked from other Java virtual machines, possibly on different hosts.

repudiation. Backing out of, or denying taking part in, an e-business transaction.

RMI. See *Remote Method Invocation*.

role-based desktop. A view rendered in a user's browser that provides access to the services the user is authorized to access when they log on to a solution. For example, a user with a defined role of buyer typically has authorization to create and view purchase orders, or a user with a role of administrator typically has authorization to add users and change passwords. In Business Integrator access to the specific services for each user is rendered in the user's browser.

RosettaNet. A non-profit organization that seeks to implement standards for supply chain management transactions on the Internet. The group includes companies such as Microsoft, Netscape, and IBM, and is working to standardize labels for elements such as product descriptions, part numbers, pricing data, and inventory status. The group aims to implement many of its goals through *XML*.

RosettaNet channel. A type of *channel* in Partner Agreement Manager.

S

SCM. See *Supply Chain Management*

SecureWay Directory. A Lightweight Directory Access Protocol (LDAP) cross-platform, highly scalable, robust directory server for security and e-business solutions.

SecureWay Policy Director. A Tivoli product that provides highly available, centralized, authentication, authorization, and user management.

SecureWay Policy Director WebSEAL. The authentication component of SecureWay Policy Director, the product that provides highly available, centralized, authentication, authorization, and user management.

sender channel. A channel that moves messages from the source to the target machine.

service. In the Business Integrator programming model, a set of command operations that is exposed within a public process or a private process.

servlet. An application program, written in the Java programming language, that is executed on a Web server. A reference to a servlet appears in the markup for a Web page, in same way that a reference to a graphics file appears. The Web server executes the servlet and sends the results of the execution (if there are any) to the Web browser.

session bean. An enterprise bean that is created by a client, that usually exists only for the duration of a single client/server session, and which is responsible for managing processes and tasks. A session bean may be transactional, but it is not recoverable if a system crash occurs. Session bean objects can be either stateless or they can maintain conversational state across methods and transactions. Contrast with *entity bean*.

solution. The realization of a business process. The solution is an instance of a solution template. At build time, you build a solution template and employ the project to organize and contain its artifacts. Before deployment the solution is published to create a solution package, and at run time, solutions are deployed, sometimes customized for the run time environment, and then run. When it is running in production, the solution is key to Business Integrators added value.

See *business process, solution template, artifacts. solution package and deployment application*.

solution artifact. See *artifact*.

Solution Console. An extensible application providing a unified view of selected aspects of a solution and solution services at the application level. Solution Console is a web-enabled interface that provides access to data that is persisted in the *Audit Log server*, the *Exception Management server*, and *Trace server* databases.

Solution Deployment Wizard. A wizard called from the Platform Console that invokes the *deployment application* to deploy a *solution package* onto the runtime system. The wizard also allows the redeployment of solution packages that have already been deployed, or partly deployed.

solution elements. The component parts of a solution. An element is a group of artifacts that is installed on one machine. Typical elements of solutions include: Business Processes Model, System Registry, Business Objects, application adapters, Business Flow, Web Clients, and Program Clients.

Solution Manager. The software that provides the infrastructure for monitoring and managing the Business Integrator system. This includes the *Platform Console, Product Console Launchpad* and *Solution Console*, which are used to manage the Business Integrator system and solution.

Solution Manager Client. Software that is installed with most of the facilities of Business Integrator, and which provides the solution management and deployment frameworks. These frameworks consist of *Java Management Extensions (JMX)* and *Managed Beans (MBeans)* to support the management of solutions and deployment of *artifacts*.

solution package. A zip file comprising solution elements, metadata, and scripts created by the publishing process of Solution Studio. The metadata contains information that determines how solution elements will map to machines and facilities in the runtime system. The scripts are used by the *deployment application* in the deployment of the solution to the runtime system.

Solution Services. Services provided by Solution Manager that include audit logging, tracing, events, and exception management.

Solution Studio. The Business Integrator component that is used to define the business processes and assemble the solution.

Solution Studio wizards. Wizards provided by Solution Studio for use in developing solutions. The wizards perform a variety of tasks. For example, many of the tools (products such as MQSeries Adapter Builder) that you use to create the solution *artifacts* are launched by using Solution Studio wizards. These wizards are also used to store the created artifacts in the project folders in the *ClearCase repository*. Solution Studio provides online help for using the wizards.

solution template. In build time, a representation of the business process that the solution is intended to carry out later at run time. In a business domain, such as supply chain management, the solution template defines the following:

- A business process model that defines the business process, for example, in the areas of enterprise integration, planning, forecasting, replenishment, scheduling, order management, and transportation. A typical business process is “purchase order”.
- A system registry that defines the run-time repository that contains system configuration and solution-specific information such as users, roles, and trading partner profiles.

- Business objects, which define the distributed objects necessary to support the business process.
- Adapters, which define the connection between gateways, the Business Flow Manager and Endpoints.
- Business flow, which provides a detailed view of the business process.

You model and create the solution template through Solution Studio, and publish the solution template as a solution in a solution package.

Solution templates typically can be reusable assets, and are key to Business Integrator's added value.

solution topology view. A view of a topology that shows a tree structure in terms of the elements and artifacts that make up a solution instance. See *topology*, *facility* and *solution*.

Standard Bean. A class that implements its own MBean interface, See *Managed Bean* and contrast with *Dynamic MBean.*

stateful session bean. A *session bean* that has a conversational state.

state machine. Software that defines one or more states with multiple transitions. Each transition contains a to-state, transition event, conditions, and a set of actions (or commands). For the transition to go to the next state (the to-state), it must successfully execute all the actions defined for the current state. In Business Integrator, a state machine can be realized in a *microflow*.

stateless session bean. A *session bean* that has no conversational state. All instances of a stateless session bean are identical.

Supply Chain Management (SCM). The management of resources, functions, and sequence of processes used by organizations involved in the supply of raw materials and products, and their delivery to manufacturers, wholesalers, retailers, and finally consumers. Business Integrator provides SCM in terms of integrated design, development, and deployment tools for creating solutions that manage the supply of goods and services between supplier and consumer.

T

task. A step in the business process.

topology. A definition of the arrangement of physical machines, together with the software products and components installed on these machines, that make up a Business Integrator runtime environment. A number of *predefined topologies* are shipped with Business Integrator, and one of these is selected at installation time.

Topology Repository. An XML file that stores the details of the machines, facilities and products that make up the topology. The Topology Repository is accessed by Business Integrator runtime components and used to display topology views.

Topology Server. The software that creates, and controls access, to the Topology Repository.

topology type. The identification of an object in the topology repository, for example, computer systems, facilities and products are topology types. Each topology type can have zero, one, or more properties, which can be displayed using the Platform Console or Product Console Launchpad.

topology view. A view of the topology in terms of either the logical, physical, or solution-related elements of the topology. You can use both the Product Console Launchpad and the Platform Console to display a *logical topology view*, *physical topology view*, or a *solution topology view*.

Trace server. The Business Integrator component that allows the storage and retrieval of trace information in a DB2 database.

trading partner agreement (TPA). The formal agreement between trading partners.

transition. A change in state when certain conditions are met.

transmission queue. A queue that stores the messages that are to be sent across a channel.

Trust and Access Manager. The Business Integrator software components that control access to the system and its associated business applications. The Trust and Access Manager provides authorization, authentication, and directory services for Business Integrator. Trust and Access Manager grants users and other business applications access, based on their authorized solutions and roles. The audit and logging capabilities of Trust and Access Manager allow administrators to monitor the system for security breaches.

TPA. See *trading partner agreement*.

U

UML. Unified Modeling Language. A general-purpose notational language for specifying and visualizing complex software, especially object-oriented projects. UML builds on previous notational methods such as Booch, OMT, and OOSE.

UNS. Short for User Name Server. See *MQSeries Integrator User Name Server*.

utilities. In the context of Solution Studio, executable software and associated documentation that can prove useful in building and running solutions. Utility software can be found in the utilities directory in Solution Studio.

W

WebDAV. An abbreviation for Web Distributed Authoring and Version. It is a set of extensions to the HTTP protocol that allows users to collaboratively edit and manage files on remote web servers.

Web Proxy Server. Software that resides in the DMZ and which performs access control on inbound messages from Web clients to Business Integrator

WebSeal. See *SecureWay Policy Director WebSEAL*.

WebSphere. A family of IBM software products that provides a development and deployment environment for basic Web publishing and for transaction-intensive, enterprise-scale e-business applications.

WebSphere Application Server. An e-business application deployment environment built on open standards-based technology. The Advanced Edition is a high-performance EJB server for implementing EJB components that incorporate business logic.

worker bean. An enterprise bean that, in concert with a JMS Listener, decides which enterprise bean in the Business Flow Manager to invoke. See *JMS Listener* for information about how they work together.

Worker beans are configured through the LDAP directory.

worker message bean. Synonymous with *worker bean*.

workflow. The sequence of activities performed in accordance with the business processes of an enterprise. For a full definition, refer to the MQSeries Workflow documentation.

X

X.500. The directory services standard of ITU, ISO, and IEC.

XML. See *Extensible Markup Language*.

XML channel. A type of *channel* in Partner Agreement Manager.

XML Metadata Interchange (XMI). A proposal from the *Object Management Group* that uses the *Extensible Markup Language* (XML) to provide a standard way for programmers and other users to exchange information about metadata (essentially, information about what a set of data consists of and how it is organized). XMI is intended to help programmers using the *Unified Modeling Language* with different languages and development tools to exchange their data models with each other. XMI can also be used to exchange information about data warehouses.

The XMI format standardizes the way in which any set of metadata is described and requires users across many industries and operating environments to see data the same way.

Bibliography

This bibliography lists the books in the IBM WebSphere Business Integrator and associated libraries.

IBM WebSphere Business Integrator library

The Business Integrator library consists of the following books:

- *WebSphere Business Integrator Concepts and Planning, GC34-5960*

  This book introduces the Business Integrator system, providing a high-level system overview, defining the system capabilities, and describing its value to e-businesses. This book also provides the information that you need to plan the installation of Business Integrator.

- *WebSphere Business Integrator Installation Guide for Windows NT, GC34-5961*

  This book is a guide to installing and configuring Business Integrator, It contains information about:

  - Selecting your required topology
  - Installing and configuring the base products and software components of Business Integrator on each machine in the topology
  - Installing and configuring firewalls and proxies

- *WebSphere Studio Business Integrator Extensions Installation Guide, SC34-5962*

  This book is a guide to installing and configuring Solution Studio, It also contains information about setting up clients and servers, and creating projects.

- *WebSphere Business Integrator Run Time*

  This book is a comprehensive guide to the Business Integrator runtime system, providing the following information:

  - Detailed conceptual information about the runtime components of Business Integrator.
  - Deployment of solutions to the runtime system
  - System administration, such as starting and stopping software components and base products, defining users, and using the Exception Console.
  - General problem determination information, including how to trace and debug, and information on obtaining help from technical support

- *WebSphere Business Integrator Messages*

  This book lists the error messages that are produced by Business Integrator and provides references to the documentation for the messages of base products.
- *WebSphere Studio Business Integrator Extensions Developer's Guide*

  This book describes how to create a Business Integrator solution, beginning with the solution design phase, to the solution implementation phase, and finally the solution deployment phase using a sample business problem. This book also provides procedures for assembling a Business Integrator solution in the run-time environment and a description of how to use the Solution Studio for solution design and implementation.
- *WebSphere Business Integrator DataInterchange for Windows NT User's Guide, SC34-5963*

  This book is a guide to installing and using DataInterchange, in the Business Integrator environment.
- *WebSphere Business Integrator Solution Samples,*

  This book discusses the two sample templates provided with Business Integrator and Solution Studio, the user-registration sample and the purchase-order management sample. It provides instructions for developing, deploying, and running the samples; it also discusses the programming model for Business Integrator Version 2.*x*.
- *WebSphere Business Integrator Process Broker Services Installation and Configuration Guide,*

  This book explains how to install and configure Process Broker Services.
- *WebSphere Business Integrator Process Broker Services Concepts Guide,*

  This book introduces the concepts involved in the Process Broker Services component of the Business Integrator system. This book also includes information on a sample that uses Process Broker Services.
- *WebSphere Business Integrator Process Broker Services Developer's Guide,*

  This book explains how to use the Application Programming Interfaces provided by Process Broker Services to create and build solution artifacts. This book also provides code samples for many of these interfaces to enable developers to understand how to implement the interfaces provided by Process Broker Services.
- *WebSphere Business Integrator Data Access Object Utility Installation and User's Reference,*

  This book describes how to install the Data Acess Object utility and explains the concept of using XML to represent SQL queries for data retrieval. This book is for solution developers who want to use XML to create database queries.

You can find the latest versions of the books at the following Web site:

`http://www-4.ibm.com/software/webservers/btobintegrator/`

This site contains links to the Web sites of the underlying products of IBM WebSphere Business Integrator.

Related documentation

WebSphere Business Integrator also provides a number of external application programming interfaces (API). HTML documentation that is generated using the Javadoc tool is provided for these APIs. For a list of the APIs, refer to the *WebSphere Business Integrator Run Time* book.

WebSphere Partner Agreement Manager library

The Partner Agreement Manager Version 2 Release 2 library consists of:

- *Partner Agreement Manager Installation Guide*, GC34-5964
- *Partner Agreement Manager Administrator's Guide*
- *Partner Agreement Manager User's Guide*
- *Partner Agreement Manager Adapter Developer's Guide*
- *Partner Agreement Manager Script Developer's Guide*
- *Partner Agreement Manager External API Guide*
- *Partner Agreement Manager Adapters for MQSeries User's Guide*
- *Partner Agreement Manager Channel Toolkit Configuration Guide*
- *Partner Agreement View User's Guide*, GC34-5965
- *B2B Alliance Manager iForms User's Guide*,
- *WebSphere Partner Agreement Manager Business Process Integration Adapter Guide*.

DataInterchange library

The DataInterchange Version 3 Release 1 library consists of:

- *DataInterchange Client User's Guide*, SB34-2010
- *DataInterchange Administrator's Guide*, SB34-2002
- *DataInterchange Installation Guide*, GB09-8070
- *DataInterchange Messages and Codes*, SB34-2000
- *DataInterchange Programmer's Reference*, SB34-2001

Other Libraries

You can find important information in the libraries of the following products:

- DB2® UDB
  - *IBM DB2 Universal Database Quick Beginnings Version 6.1* , S10J-8149
- MQSeries®
  - *MQSeries for Windows NT Quick Beginnings*, GC34-5389
  - *MQSeries System Administration*, SC33-1873
  - *MQSeries Using Java*, SC34-5456
  - *MQSeries MQSC Command Reference*, SC33-1369
  - *MQSeries Queue Manager Clusters*, SC34-5349
  - *MQSeries Integrator Introduction and Planning*, GC24-5599
  - *MQSeries Integrator for Windows NT Installation* , GC34-5600
  - *MQSeries Workflow Getting Started with Buildtime*, SH12-6286

  - *MQSeries Workflow Getting Started with Runtime*, SH12-6287
  - *MQSeries Adapter Kernel for Multiplatforms: Quick Beginnings*, GC34-5855
  - *MQSeries Adapter Kernel for Multiplatforms: Problem Determination Guide*, GC34-5897
  - *MQSeries Adapter Builder for Windows NT: Using the Control Center*, GC34-5882
- SecureWay®
  - *SecureWay Policy Director Up and Running*, SCT6-3KNA
  - *SecureWay Policy Director Base Administration Guide*
  - *SecureWay Firewall User's Guide*, CG31-8658
- VisualAge®
  - *VisualAge Java, Enterprise Edition Getting Started*
  - *VisualAge C++ Professional for Windows NT Getting Started*
- WebSphere™ Application Server
  - *Introduction to WebSphere Application Server*, SC09-4430

Index

Special Characters

A

B

C

D

E

F

G

H

I

J

K

L

M

N

O

P

R

S

T

U

V

W

X

File Number: BIZAAN00

Printed in U.S.A.