WebSphere® Process Server for z/OS

Version 6.0.2

Developing and Deploying Modules

Note
Before using this information, be sure to read the general information in "Notices" on page 43.

30 March 2007

This edition applies to version 6, release 0, modification 2 of WebSphere Process Server for z/OS (product number 5655-N53) and to all subsequent releases and modifications until otherwise indicated in new editions.

To send us your comments about this document, email doc-comments@us.ibm.com. We look forward to hearing from you.

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

© Copyright International Business Machines Corporation 2006, 2007. All rights reserved.
US Government Users Restricted Rights – Use, duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp.

Contents

© Copyright IBM Corp. 2006, 2007

Chapter 1. Overview of developing modules

A module is a basic deployment unit for a WebSphere Process Server application. A module contains one or more component libraries and staging modules used by the application. A component may reference other service components. Developing modules involves ensuring that the components, staging modules, and libraries (collections of artifacts referenced by the module) required by the application are available on the production server.

WebSphere® Integration Developer is the main tool for developing modules for deployment to WebSphere Process Server. Although you can develop modules in other environments, it is best to use WebSphere Integration Developer.

WebSphere Process Server supports two types of service modules: modules for business services and mediation modules. A module for business services implements the logic of a process. A mediation module allows communication between applications by transforming the service invocation to a format understood by the target, passing the request to the target and returning the result to the originator.

The following sections address how to implement and update modules on WebSphere Process Server.

A synopsis on components

A component is the basic building block to encapsulate reusable business logic. A service component is associated with interfaces, references and implementations. The interface defines a contract between a service component and a calling component. With WebSphere Process Server, a service module can either export a service component for use by other modules or import a service component for use. To invoke a service component, a calling module references the interface to the service component. The references to the interfaces are resolved by configuring the references from the calling module to their respective interfaces.

To develop a module you must do the following activities:

1. Define interfaces for the components in the module
2. Define, modify, or manipulate business objects used by service components
3. Define or modify service components through its interfaces.

   Note: A service component is defined through its interface.
4. Optionally, export or import service components.
5. Create an EAR file you use to install a module that uses components. You create the file using either the export EAR feature in WebSphere Integration Developer or the serviceDeploy command to create an EAR file to install a service module that uses service components.

Development types

WebSphere Process Server provides a component programming model to facilitate a service-oriented programming paradigm. To use this model, a provider exports interfaces of a service component so that a consumer can import those interfaces and use the service component as if it were local. A developer uses either

© Copyright IBM Corp. 2006, 2007

strongly-typed interfaces or dynamically-typed interfaces to implement or invoke the service component. The interfaces and their methods are described in the References section within this information center.

After installing service modules to your servers, you can use the administrative console to change the target component for a reference from an application. The new target must accept the same business object type and perform the same operation that the reference from the application is requesting.

Service component development considerations

When developing a service component, ask yourself the following questions:

- Will this service component be exported and used by another module?

  If so, make sure the interface you define for the component can be used by another module.
- Will the service component take a relatively long time to run?

  If so, consider implementing an asynchronous interface to the service component.
- Is it beneficial to decentralize the service component?

  If so, consider having a copy of the service component in a service module that is deployed on a cluster of servers to benefit from parallel processing.
- Does your application require a mixture of 1-phase and 2-phase commit resources?

  If so, make sure you enable last participant support for the application.

  Note: If you create your application using WebSphere Integration Developer or create the installable EAR file using the serviceDeploy command, these tools automatically enable the support for the application. See the topic, "Using one-phase and two-phase commit resources in the same transaction" in the WebSphere Application Server for z/OS information center.

Developing service modules

A service component must be contained within a service module. Developing service modules to contain service components is key to providing services to other modules.

This task assumes that an analysis of requirements shows that implementing a service component for use by other modules is beneficial.

After analyzing your requirements, you might decide that providing and using service components is an efficient way to process information. If you determine that reusable service components would benefit your environment, create a service module to contain the service components.

1. Identify service components other modules can use.

   Once you have identified the service components, continue with Developing service components.
2. Identify service components within an application that could use service components in other service modules.

   Once you have identified the service components and their target components, continue with Invoking components.
3. Connect the client components with the target components through wires.

Developing service components

Develop service components to provide reusable logic to multiple applications within your server.

This task assumes that you have already developed and identified processing that is useful for multiple modules.

Multiple modules can use a service component. Exporting a service component makes it available to other modules that refer to the service component through an interface. This task describes how to build the service component so that other modules can use it.

Note: A single service component can contain multiple interfaces.

1. Define the data object to move data between the caller and the service component.

   The data object and its type is part of the interface between the callers and the service component.
2. Define an interface that the callers will use to reference the service component.

   This interface definition names the service component and lists any methods available within the service component.
3. Develop the class that defines the implementation.
   - If the component is long running (or asynchronous), continue with step 4.
   - If the component is not long running (or synchronous), continue with step 5.
4. Develop an asynchronous implementation.

   Important: An asynchronous component interface cannot have a joinsTransaction property set to `true`.

   a. Define the interface that represents the synchronous service component.

   b. Define the implementation of the service component.

   c. Continue with step 6.
5. Develop a synchronous implementation.

   a. Define the interface that represents the synchronous service component.

   b. Define the implementation of the service component.
6. Save the component interfaces and implementations in files with a .java extension.
7. Package the service module and necessary resources in a JAR file.

   See “Deploying a module to a production server” in this information center for a description of steps 7 through 9.
8. Run the serviceDeploy command to create an installable EAR file containing the application.
9. Install the application on the server node.
10. **Optional:** Configure the wires between the callers and the corresponding service component, if calling a service component in another service module.

    The “Administering” section of this information center describes configuring the wires.

Examples of developing components

This example shows a synchronous service component that implements a single method, CustomerInfo. The first section defines the interface to the service component that implements a method called getCustomerInfo.

```
public interface CustomerInfo {
 public Customer getCustomerInfo(String customerID);
}
```

The following block of code implements the service component.

```
public class CustomerInfoImpl implements CustomerInfo {
 public Customer getCustomerInfo(String customerID) {
  Customer cust = new Customer();

  cust.setCustNo(customerID);
  cust.setFirstName("Victor");
  cust.setLastName("Hugo");
  cust.setSymbol("IBM");
  cust.setNumShares(100);
  cust.setPostalCode(10589);
  cust.setErrorMsg("");

  return cust;
 }
}
```

This example develops an asynchronous service component. The first section of code defines the interface to the service component that implements a method called getQuote.

```
public interface StockQuote {

 public float getQuote(String symbol);
}
```

The following section is the implementation of the class associated with StockQuote.

```
public class StockQuoteImpl implements StockQuote {

 public float getQuote(String symbol) {


     return 100.0f;
 }
}
```

This next section of code implements the asynchronous interface, StockQuoteAsync.

```
public interface StockQuoteAsync {

 // deferred response
 public Ticket getQuoteAsync(String symbol);
 public float getQuoteResponse(Ticket ticket, long timeout);

 // callback
 public Ticket getQuoteAsync(String symbol, StockQuoteCallback callback);
}
```

This section is the interface, StockQuoteCallback, which defines the onGetQuoteResponse method.

```
public interface StockQuoteCallback {

 public void onGetQuoteResponse(Ticket ticket, float quote);
}
```

Invoke the service.

Invoking components

Components with modules can use components on any node of a WebSphere Process Server cluster.

Before invoking a component, make sure that the module containing the component is installed on WebSphere Process Server.

Components can use any service component available within a WebSphere Process Server cluster by using the name of the component and passing the data type the component expects. Invoking a component in this environment involves locating and then creating the reference to the required component.

Note: A component in a module can invoke a component within the same module, known as an intra-module invocation. Implement external calls (inter-module invocations) by exporting the interface in the providing component and importing the interface in the calling component.

Important: When invoking a component that resides on a different server than the one on which the calling module is running, you must perform additional configurations to the servers. The configurations required depend on whether the component is called asynchronously or synchronously. How to configure the application servers in this case is described in related tasks.

1. Determine the components required by the calling module.

   Note the name of the interface within a component and the data type that interface requires.
2. Define a data object.

   Although the input or return can be a Java™ class, a service data object is optimal.
3. Locate the component.
   a. Use the ServiceManager class to obtain the references available to the calling module.
   b. Use the locateService() method to find the component.

      Depending on the component, the interface can either be a Web Service Descriptor Language (WSDL) port type or a Java interface.
4. Invoke the component either synchronously or asynchronously.

   You can either invoke the component through a Java interface or use the invoke() method to dynamically invoke the component.
5. Process the return.

   The component might generate an exception, so the client has to be able to process that possibility.

Example of invoking a component

The following example creates a ServiceManager class.

```
ServiceManager serviceManager = new ServiceManager();
```

The following example uses the ServiceManager class to obtain a list of components from a file that contains the component references.

```
InputStream myReferences = new FileInputStream("MyReferences.references");
ServiceManager serviceManager = new ServiceManager(myReferences);
```

The following code locates a component that implements the StockQuote Java interface.

```
StockQuote stockQuote = (StockQuote)serviceManager.locateService("stockQuote");
```

The following code locates a component that implements either a Java or WSDL port type interface. The calling module uses the Service interface to interact with the component.

Tip: If the component implements a Java interface, the component can be invoked through either the interface or the invoke() method.

```
Service stockQuote = (Service)serviceManager.locateService("stockQuote");
```

The following example shows MyValue, code that calls another component.

```
public class MyValueImpl implements MyValue {

 public float myValue throws MyValueException {

  ServiceManager serviceManager = new ServiceManager();

     // variables
         Customer customer = null;
         float quote = 0;
         float value = 0;

     // invoke
         CustomerInfo cInfo =
      (CustomerInfo)serviceManager.locateService("customerInfo");
         customer = cInfo.getCustomerInfo(customerID);

     if (customer.getErrorMsg().equals("")) {

         // invoke
       StockQuoteAsync sQuote =
      (StockQuoteAsync)serviceManager.locateService("stockQuote");
        Ticket ticket =  sQuote.getQuoteAsync(customer.getSymbol());
    // ... do something else ...
       quote =  sQuote.getQuoteResponse(ticket, Service.WAIT);

         // assign
          value = quote * customer.getNumShares();
     } else {

         // throw
         throw new MyValueException(customer.getErrorMsg());
     }
     // reply
         return value;
 }
}
```

Configure the wires between the calling module references and the component interfaces.

Dynamically invoking a component

When an module invokes a component that has a Web Service Descriptor Language (WSDL) port type interface, the module must invoke the component dynamically using the invoke() method.

This task assumes that a calling component is invoking a component dynamically.

With a WSDL port type interface, a calling component must use the invoke() method to invoke the component. A calling module can also invoke a component that has a Java interface this way.

1. Determine the module that contains the component required.
2. Determine the array required by the component.

   The input array can be one of three types:
   - Primitive uppercase Java types or arrays of this type
   - Ordinary Java classes or arrays of the classes
   - Service Data Objects (SDOs)
3. Define an array to contain the response from the component.

   The response array can be of the same types as the input array.
4. Use the invoke() method to invoke the required component and pass the array object to the component.
5. Process the result.

Examples of dynamically invoking a component

In the following example, a module uses the invoke() method to call a component that uses primitive uppercase Java data types.

```
Service service = (Service)serviceManager.locateService("multiParamInf");

  Reference reference = service.getReference();

  OperationType methodMultiType =
    reference.getOperationType("methodWithMultiParameter");

  Type t = methodMultiType.getInputType();

  BOFactory boFactory = (BOFactory)serviceManager.locateService
    ("com/ibm/websphere/bo/BOFactory");

  DataObject paramObject = boFactory.createbyType(t);

  paramObject.set(0,"input1")
  paramObject.set(1,"input2")
  paramObject.set(2,"input3")

  service.invoke("methodMultiParamater",paramObject);
```

The following example uses the invoke method with a WSDL port type interface as the target.

```
Service serviceOne = (Service)serviceManager.locateService("multiParamInfWSDL");

 DataObject dob = factory.create("http://MultiCallWSServerOne/bos", "SameBO");
   dob.setString("attribute1", stringArg);

 DataObject wrapBo = factory.createByElement
  ("http://MultiCallWSServerOne/wsdl/ServerOneInf", "methodOne");
   wrapBo.set("input1", dob); //wrapBo encapsulates all the parameters of methodOne
   wrapBo.set("input2", "XXXX");
```

```
  wrapBo.set("input3", "yyyy");

DataObject resBo= (DataObject)serviceOne.invoke("methodOne", wrapBo);
```

Overview of isolating modules and targets

When developing modules, you will identify services that multiple modules can use. Leveraging services this way minimizes your development cycle and costs. When you have a service used by many modules, you should isolate the invoking modules from the target so that if the target is upgraded, switching to the new service is transparent to the calling module. This topic contrasts the simple invocation model and the isolated invocation model and provides an example of how isolation can be useful. While describing a specific example, this is not the only way to isolate modules from targets.

Simple invocation model

While developing a module, you might use services that are located in other modules. You do this by importing the service into the module and then invoking that service. The imported service is "wired" to the service exported by the other module either in WebSphere Integration Developer or by binding the service in the administrative console. Simple invocation model illustrates this model.

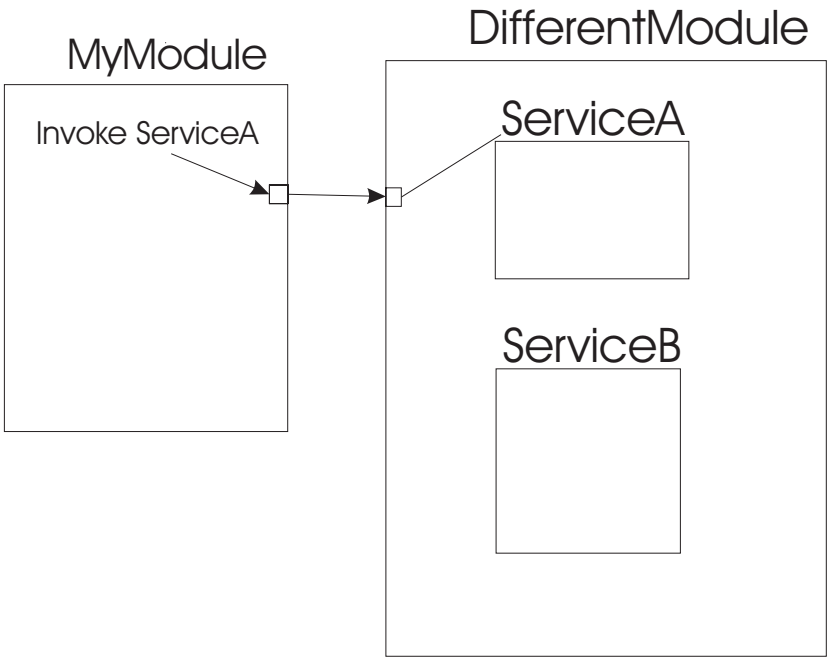

Figure 1. Simple invocation model

Isolated invocation model

To change the target of an invocation without stopping invoking modules, you can isolate the invoking modules from the target of the invocation. This allows the modules to continue processing while you change the target because you are not changing the module itself but the downstream target. Example of isolating applications shows how isolation allows you to change the target without affecting the status of the invoking module.

Example of isolating applications

Using the simple invocation model, multiple modules invoking the same service would look much like Multiple applications invoking a single service . MODA, MODB, and MODC all invoke CalculateFinalCost.

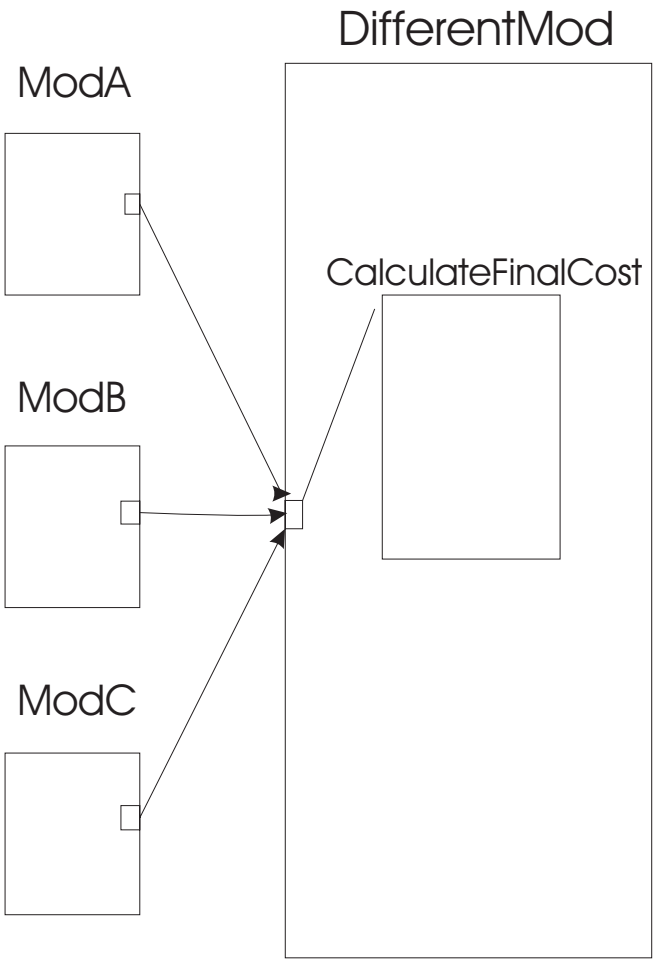

Figure 2. Multiple applications invoking a single service

The service provided by CalculateFinalCost needs updating so that new costs are reflected in all modules that use the service. The development team builds and tests a new service UpdatedCalculateFinal to incorporate the changes. You are ready to bring the new service into production. Without isolation, you would have to update all of the modules invoking CalculateFinalCost to invoke UpdateCalculateFinal. With isolation, you only have to change the binding that connects the buffer module to the target.

Note: Changing the service this way allows you to continue to provide the original service to other modules that may need it.

Using isolation, you create a buffer module between the applications and the target (see Isolated invocation model invoking UpdateCalculateFinal).

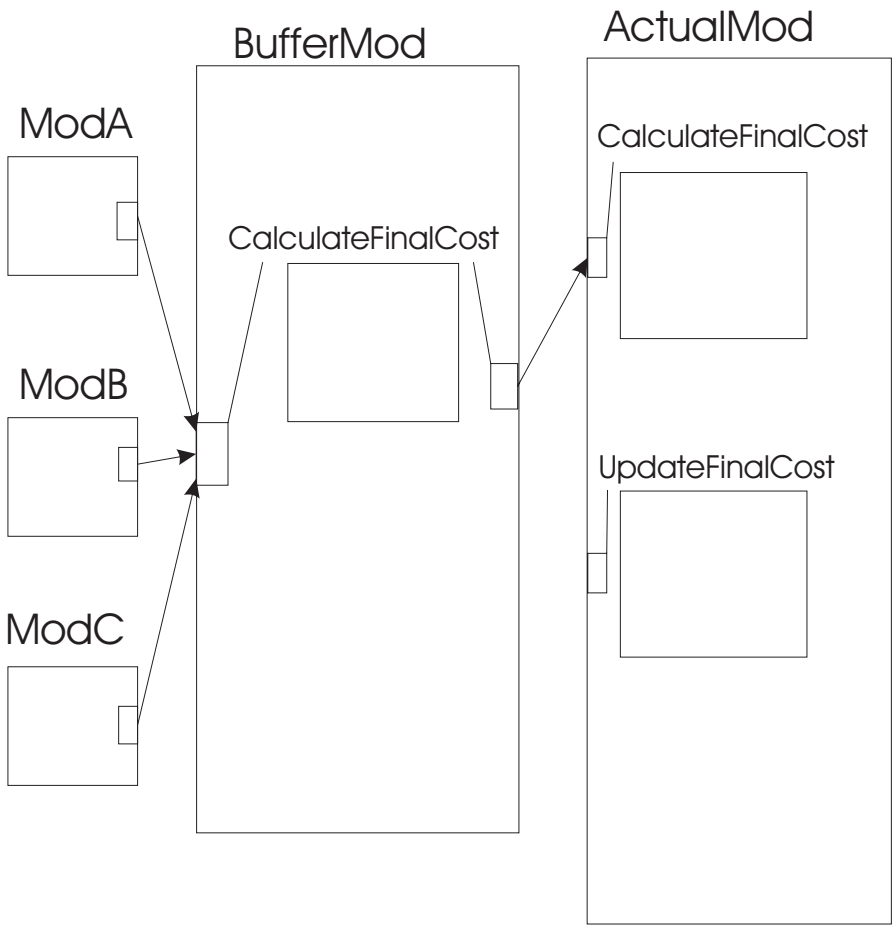

Figure 3. Isolated invocation model invoking UpdateCalculateFinal

With this model, the invoking modules do not change, you just have to change the binding from the buffer module import to the target (see Isolated invocation model invoking UpdatedCalculateFinal).

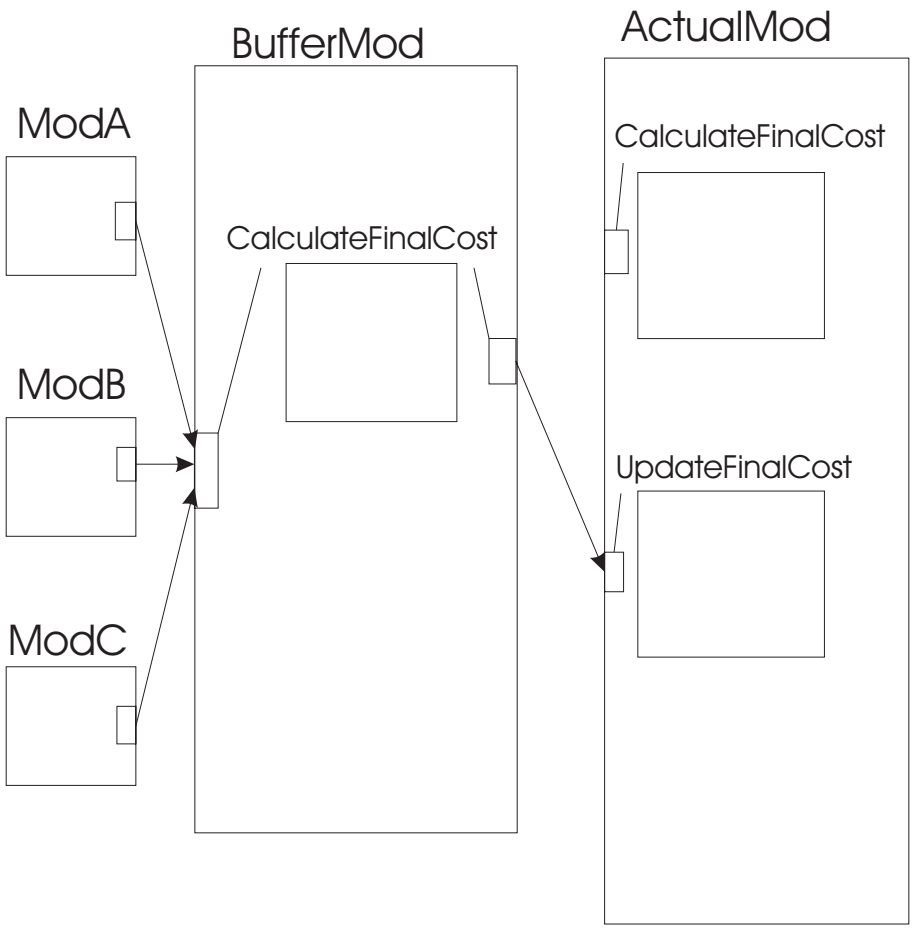

Figure 4. Isolated invocation model invoking UpdatedCalculateFinal

If the buffer module invokes the target synchronously, when you restart the buffer module (whether a mediation module or a service for business module) the results returned to the original application come from the new target. If the buffer module invokes the target asynchronously, the results returned to the original application come from the new target on the next invocation.

Chapter 2. Overview of preparing and installing modules

Installing modules (also known as deploying) activates the modules in either a test environment or a production environment. This overview briefly describes the test and production environments and some of the steps involved in installing modules.

Note: The process for installing applications in a production environment is similar to the process described in "Developing and deploying applications" in the WebSphere Application Server for z/OS information center. If you are unfamiliar with those topics, review those first.

Before installing a module to a production environment, always verify changes in a test environment. To install modules to a test environment, use WebSphere Integration Developer (see the WebSphere Integration Developer information center for more information). To install modules to a production environment, use WebSphere Process Server.

This topic describes the concepts and tasks needed to prepare and install modules to a production environment. Other topics describe the files that house the objects that your module uses and help you move your module from your test environment into your production environment. It is important to understand these files and what they contain so you can be sure that you have correctly installed your modules.

Libraries and JAR files overview

Modules often use artifacts that are located in libraries. Artifacts and libraries are contained in Java archive (JAR) files that you identify when you deploy a module.

While developing a module, you might identify certain resources or components that could be used by various pieces of the module. These resources or components could be objects that you created while developing the module or already existing objects that reside in a library that is already deployed on the server. This topic describes the libraries and files that you will need when you install an application.

What is a library?

A library contains objects or resources used by multiple modules within WebSphere Integration Developer. The artifacts can be in JAR, resource archive (RAR), or Web service archive (WAR) files. Some of these artifacts include:

- Interfaces or Web services descriptors (files with a .wsdl extension)
- Business object XML schema definitions (files with an .xsd extension)
- Business object maps (files with a .map extension)
- Relationship and role definitions (files with a .rel and .rol extension)

When a module needs an artifact, the server locates the artifact from the EAR class path and loads the artifact, if it is not already loaded, into memory. From that point on, any request for the artifact uses that copy until it is replaced. Figure 5 on page 14 shows how an application contains components and related libraries.

© Copyright IBM Corp. 2006, 2007

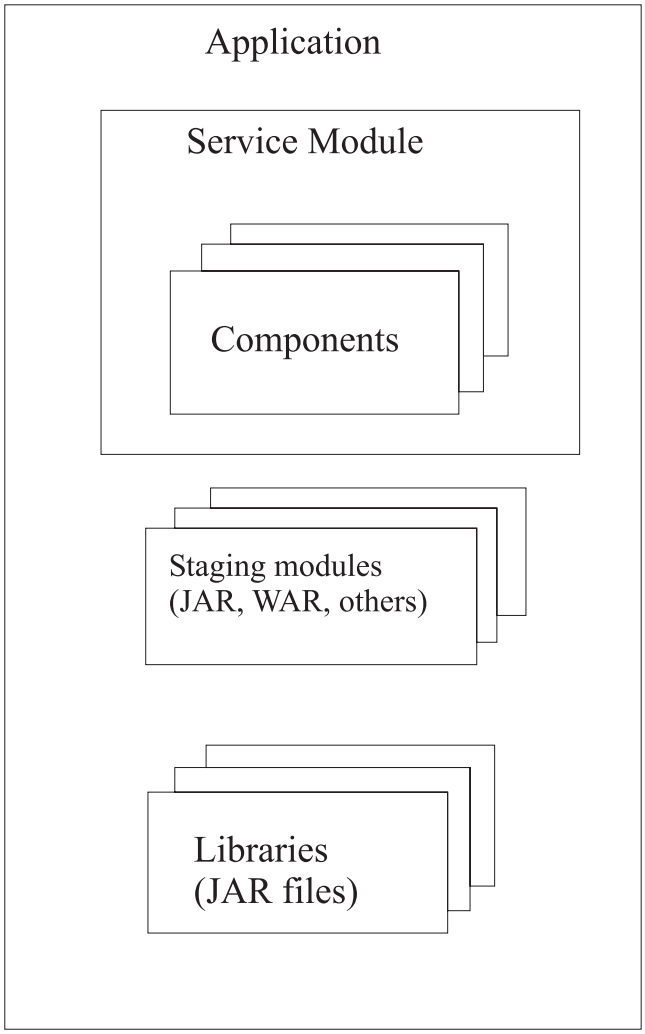

Figure 5. Relationship amongst module, component and library

What are JAR, RAR, and WAR files?

There are a number of files that can contain components of a module. These files are fully described in the Java Platform, Enterprise Edition specification. Details about JAR files can be found in the JAR specification.

In WebSphere Process Server, a JAR file also contains an application, which is the assembled version of the module with all the supporting references and interfaces to any other service components used by the module. To completely install the application, you need this JAR file, any other libraries such as JAR files, Web services archive (WAR) files, resource archive (RAR) files, staging libraries (Enterprise Java Beans - EJB) JAR files, or any other archives, and create an installable EAR file using the serviceDeploy command (see Installing a module on a production server).

Naming conventions for staging modules

Within the library, there are requirements for the names of the staging modules. These names are unique for a specific module. Name any other modules required to deploy the application so that conflicts with the staging module names do not occur. For a module named *myService*, the staging module names are:

- *myService*App
- *myService*EJB
- *myService*EJBClient
- *myService*Web

Note: The serviceDeploy command only creates the *myService* Web staging module if the service includes a WSDL port type service.

Considerations when using libraries

Using libraries provides consistency of business objects and consistency of processing amongst modules because each calling module has its own copy of a specific component. To prevent inconsistencies and failures it is important to make sure that changes to components and business objects used by calling modules are coordinated with all of the calling modules. Update the calling modules by:

1. Copying the module and the latest copy of the libraries to the production server
2. Rebuilding the installable EAR file using the serviceDeploy command
3. Stopping the running application containing the calling module and reinstall it
4. Restarting the application containing the calling module

EAR file overview

An EAR file is a critical piece in deploying a service application to a production server.

An enterprise archive (EAR) file is a compressed file that contains the libraries, enterprise beans, and JAR files that the application requires for deployment.

You create a JAR file when you export your application modules from WebSphere Integration Developer. Use this JAR file and any other artifact libraries or objects as input to the installation process. The serviceDeploy command creates an EAR file from the input files that contain the component descriptions and Java code that comprise the application.

Preparing to deploy to a server

After developing and testing a module, you must export the module from a test system and bring it into a production environment for deployment. To install an application you also should be aware of the paths needed when exporting the module and any libraries the module requires.

Before beginning this task, you should have developed and tested your modules on a test server and resolved problems and performance issues.

This task verifies that all of the necessary pieces of an application are available and packaged into the correct files to bring to the production server.

Note: You can also export an enterprise archive (EAR) file from WebSphere Integration Developer and install that file directly into WebSphere Process Server.

Important: If the services within a component use a database, install the application on a server directly connected to the database.

1. Locate the folder that contains the components for the module you are to deploy.

   The component folder should be named *module-name* with a file in it named *module*.module, the base module.
2. Verify that all components contained in the module are in component subfolders beneath the module folder.

   For ease of use, name the subfolder similar to *module/component*.
3. Verify that all files that comprise each component are contained in the appropriate component subfolder and have a name similar to *component-file-name*.component.

   The component files contain the definitions for each individual component within the module.
4. Verify that all other components and artifacts are in the subfolders of the component that requires them.

   In this step you ensure that any references to artifacts required by a component are available. Names for components should not conflict with the names the serviceDeploy command uses for staging modules. See Naming conventions for staging modules.
5. Verify that a references file, *module*.references, exists in the module folder of step 1.

   The references file defines the references and the interfaces within the module.
6. Verify that a wires file, *module*.wires, exists in the component folder.

   The wires file completes the connections between the references and the interfaces within the module.
7. Verify that a manifest file, *module*.manifest, exists in the component folder.

   The manifest lists the module and all the components that comprise the module. It also contains a classpath statement so that the serviceDeploy command can locate any other modules needed by the module.
8. Create a compressed file or a JAR file of the module as input to the serviceDeploy command that you will use to prepare the module for installation to the production server.

Example folder structure for MyValue module prior to deployment

The following example illustrates the directory structure for the module MyValueModule, which is made up of the components MyValue, CustomerInfo, and StockQuote.

```
MyValueModule
   MyValueModule.manifest
   MyValueModule.references
   MyValueModule.wiring
   MyValueClient.jsp
process/myvalue
   MyValue.component
   MyValue.java
   MyValueImpl.java
service/customerinfo
   CustomerInfo.component
   CustomerInfo.java
   Customer.java
   CustomerInfoImpl.java
service/stockquote
   StockQuote.component
```

```
StockQuote.java
StockQuoteAsynch.java
StockQuoteCallback.java
StockQuoteImpl.java
```

Install the module onto the production systems as described in Installing a module on a production server.

Considerations for installing service applications on clusters

Installing a service application on a cluster places additional requirements on you. It is important that you keep these considerations in mind as you install any service applications on a cluster.

Clusters can provide many benefits to your processing environment by providing economies of scale to help you balance request workload across servers and provide a level of availability for clients of the applications. Consider the following before installing an application that contains services on a cluster:

- Will users of the application require the processing power and availability provided by clustering?

  If so, clustering is the correct solution. Clustering will increase the availability and capacity of your applications.
- Is the cluster correctly prepared for service applications?

  You must configure the cluster correctly before installing and starting the first application that contains a service. Failure to configure the cluster correctly prevents the applications from processing requests correctly.
- Does the cluster have a backup?

  You must install the application on the backup cluster also.

Chapter 3. Installing a module on a production server

This topic describes the steps involved in taking an application from a test server and deploying it into a production environment.

Before deploying a service application to a production server, assemble and test the application on a test server. After testing, export the relevant files as described in *Preparing to deploy to a server* in the Developing and Deploying Modules PDF and bring the files to the production system to deploy. See the information centers for WebSphere Integration Developer and WebSphere Application Server for z/OS for more information.

1. Copy the module and other files onto the production server.

   The modules and resources (EAR, JAR, RAR, and WAR files) needed by the application are moved to your production environment.
2. Run the serviceDeploy command to create an installable EAR file.

   This step defines the module to the server in preparation for installing the application into production.

   a. Locate the JAR file that contains the module to deploy.

   b. Issue the command using the JAR file from the previous step as input.
3. Install the EAR file from step 2. How you install the applications depends on whether you are installing the application on a stand alone server or a server in a cell.

   Note: You can either use the administrative console or a script to install the application. See the WebSphere Application Server information center for additional information.
4. Save the configuration. The module is now installed as an application.
5. Start the application.

The application is now active and work should flow through the module.

Monitor the application to make sure the server is processing requests correctly.

Creating an installable EAR file using serviceDeploy

To install an application in the production environment, take the files copied to the production server and create an installable EAR file.

Before starting this task, you must have a JAR file that contains the module and services you are deploying to the server. See Preparing to deploy to a server for more information.

The serviceDeploy command takes a JAR file, any other dependent EAR, JAR, RAR, WAR and ZIP files and builds an EAR file that you can install on a server.

1. Locate the JAR file that contains the module to deploy.
2. Issue the command using the JAR file from the previous step as input.

   This step creates an EAR file.

   Note: Perform the following steps at an administrative console.

© Copyright IBM Corp. 2006, 2007

3. Select the EAR file to install in the administrative console of the server.
4. Click **Save** to install the EAR file.

Deploying applications using ANT tasks

This topic describes how to use ANT tasks to automate the deployment of applications to WebSphere Process Server. By using ANT tasks, you can define the deployment of multiple applications and have them run unattended on a server.

This task assumes the following:
- The applications being deployed have already been developed and tested.
- The applications are to be installed on the same server or servers.
- You have some knowledge of ANT tasks.
- You understand the deployment process.

Information about developing and testing applications is located in the WebSphere Integration Developer information center.

The reference portion of the information center for WebSphere Application Server for z/OS contains a section on application programming interfaces. ANT tasks are described in the package com.ibm.websphere.ant.tasks. For the purpose of this topic, the tasks of interest are ServiceDeploy and InstallApplication.

If you need to install multiple applications concurrently, develop an ANT task before deployment. The ANT task can then deploy and install the applications on the servers without your involvement in the process.

1. Identify the applications to deploy.
2. Create a JAR file for each application.
3. Copy the JAR files to the target servers.
4. Create an ANT task to run the ServiceDeploy command to create the EAR file for each server.
5. Create an ANT task to run the InstallApplication command for each EAR file from step 4 on the applicable servers.
6. Run the ServiceDeploy ANT task to create the EAR file for the applications.
7. Run the InstallApplication ANT task to install the EAR files from step 6.

The applications are correctly deployed on the target servers.

Example of deploying an application unattended

This example shows an ANT task contained in a file myBuildScript.xml.

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

<project name="OwnTaskExample" default="main" basedir=".">
 <taskdef name="servicedeploy"
   classname="com.ibm.websphere.ant.tasks.ServiceDeployTask" />
 <target name="main" depends="main2">
  <servicedeploy scaModule="c:/synctest/SyncTargetJAR"
   ignoreErrors="true"
   outputApplication="c:/synctest/SyncTargetEAREAR"
   workingDirectory="c:/synctest"
   noJ2eeDeploy="true"
   cleanStagingModules="true"/>
 </target>
</project>
```

This statement shows how to invoke the ANT task.

```
${WAS}/bin/ws_ant -f myBuildScript.xml
```

Tip: Multiple applications can be deployed unattended by adding additional project statements into the file.

Use the administrative console to verify that the newly installed applications are started and processing the workflow correctly.

Chapter 4. Installing business process and human task applications

You can distribute Service Component Architecture (SCA) Enterprise JavaBeans™ (EJB) modules that contain business processes or human tasks, or both, to deployment targets. A deployment target can be a server or a cluster.

Verify that the business process container or task container is installed and configured for each application server or cluster on which you want to install your application.

Before you install a business process or human task application, make sure that the following conditions are true:

- The servers on which you want to install the application are running.
- In each cluster, at least one server on which you want to install Enterprise JavaBeans modules with processes or tasks is running.

You can install business process and task applications from the administrative console, from the command line, or by running an administrative script, for example. When you run an administrative script to install a business process application or a human task application, a server connection is required. Do not use the `-conntype NONE` option as an installation option.

1. If you are installing an application on a cluster, verify that the application uses the data source that is named after the cluster.

   For example, if the application was generated using the default data source `BPEDB`, change the data source for the application to BPEDB_*cluster_name*, where *cluster_name* is the name of the cluster on which you installed the application.
2. Install the application.

All business process templates and human task templates are put into the start state. You can create process instances and task instances from these templates.

Before you can create process instances or task instances, you must start the application.

Deployment of models

When WebSphere Integration Developer or service deploy generates the deployment code for your process, the constructs in the process or task model are mapped to various Java 2 Enterprise Edition (J2EE) constructs and artifacts. All deployment code is packaged into the enterprise application (EAR) file. Each new version of a model that is to be deployed must be packaged into a new enterprise application.

When you install an enterprise application that contains business process model or human task model J2EE constructs, the model constructs are stored as *process templates* or *task templates*, as appropriate, in the Business Process Choreographer database. If the database system is not running, or if it cannot be accessed, the deployment fails. Newly installed templates are, by default, in the started state. However, the newly installed enterprise application is in the stopped state. Each installed enterprise application can be started and stopped individually.

© Copyright IBM Corp. 2006, 2007

New versions of a process template or task template have the same name, but a different valid-from attribute. You can deploy many different versions of a process template or task template, each in a different enterprise application. However, no two versions of the same process can have the same valid-from date. If you want to install different versions of the same process, specify a different valid-from date for each version. All the different process versions are stored in the database.

If you do not specify a valid-from date, the date is determined as follows:

- For a human task, the valid-from date is the date on which the application was installed.
- For a business process, the valid-from date is the date on which the process was modeled.

Deploying business process applications interactively

You can install an application interactively at runtime using the wsadmin tool and the installInteractive script. You can use this script to change settings that cannot be changed if you use the administrative console to install the application.

Perform the following steps to install business process applications interactively.

1. Start the wsadmin tool.

   In the *profile_root*/bin directory, enter `wsadmin`.
2. Install the application.

   At the wsadmin command-line prompt, enter the following command:

   ```
   $AdminApp installInteractive application.ear
   ```

   where *application.ear* is the qualified name of the enterprise archive file that contains your process application. You are prompted through a series of tasks where you can change values for the application.
3. Save the configuration changes.

   At the wsadmin command-line prompt, enter the following command:

   ```
   $AdminConfig save
   ```

   You must save your changes to transfer the updates to the master configuration repository. If a scripting process ends and you have not saved your changes, the changes are discarded.

Configuring process application data source and set reference settings

You might need to configure process applications that run SQL statements for the specific database infrastructure. These SQL statements can come from information service activities or they can be statements that you run during process installation or instance startup.

When you install the application, you can specify the following types of data sources:

- Data sources to run SQL statements during process installation
- Data sources to run SQL statements during the startup of a process instance
- Data sources to run SQL snippet activities

The data source required to run an SQL snippet activity is defined in a BPEL variable of type tDataSource. The database schema and table names that are

required by an SQL snippet activity are defined in BPEL variables of type tSetReference. You can configure the initial values of both of these variables.

You can use the wsadmin tool to specify the data sources.

1. Install the process application interactively using the wsadmin tool.
2. Step through the tasks until you come to the tasks for updating data sources and set references.

   Configure these settings for your environment. The following example shows the settings that you can change for each of these tasks.
3. Save your changes.

Example: Updating data sources and set references, using the wsadmin tool

In the **Updating data sources** task, you can change data source values for initial variable values and statements that are used during installation of the process or when the process starts. In the **Updating set references** task, you can configure the settings related to the database schema and the table names.

```
Task [24]: Updating data sources

//Change data source values for initial variable values at process start

Process name:  Test
// Name of the process template
Process start or installation time:  Process start
// Indicates whether the specified value is evaluated
//at process startup or process installation
Statement or variable:  Variable
// Indicates that a data source variable is to be changed
Data source name:  MyDataSource
// Name of the variable
JNDI name:[jdbc/sample]:jdbc/newName
// Sets the JNDI name to jdbc/newName
Task [25]: Updating set references

// Change set reference values that are used as initial values for BPEL variables

Process name:  Test
// Name of the process template
Variable:  SetRef
// The BPEL variable name
JNDI name:[jdbc/sample]:jdbc/newName
// Sets the JNDI name of the data source of the set reference to jdbc/newName
Schema name: [IISAMPLE]
// The name of the database schema
Schema prefix: []:
// The schema name prefix.
// This setting applies only if the schema name is generated.
Table name: [SETREFTAB]: NEWTABLE
// Sets the name of the database table to NEWTABLE
Table prefix: []:
// The table name prefix.
// This setting applies only if the prefix name is generated.
```

When you can install a process application on a cluster in which no servers are running

This topic explains the exceptional circumstances in which you might need to install an application on a cluster that has no running servers.

During the installation of a business process application on a server, the Java Naming and Directory Interface (JNDI) name of the data source of the corresponding business process container must be resolved. You cannot, therefore, install an application without a server connection. In a Network Deployment (ND) environment, this server is the deployment manager.

Restrictions lifted

If you want to install a business process application on a cluster in an ND environment, no server in the cluster need be running if the following conditions are true:

- The required data sources are defined at the cell level.
- The process application does not specify human tasks.

For process applications that have no human tasks, the data source lookup operation is accomplished within the namespace of the deployment manager, when a lookup operation in the namespace of the application server previously failed. If the application is successfully installed, ignore any error messages in the `SystemOut.log` file that indicate a failure of the data source lookup operation within the application server namespace.

When it will work

- The lookup operation within the deployment manager namespace is successful only if the data source JNDI name is defined at the cell level.
- If you use the wizard to configure a business process container or human task container on a stand-alone server, the data source is defined at the server level. The same is true if you use the configuration script `bpeconfig.jacl`, which is provided in the `ProcessChoreographer/config` directory of your application server installation. In this case, you must define the data source manually at the cell level and use this data source when you install the business process container.
- If you configure a business process container with the wizard on a cluster member, the data source is automatically defined at the cell level. The JNDI name is scoped by the cluster name. The same is true if you use the configuration script `bpeconfig.jacl`, which is provided in the `ProcessChoreographer/config` directory of your application server installation. In this case, you do not need to change anything manually.

When it will not work

Process applications that contain human tasks require an additional JNDI name lookup operation to locate the staff plug-in provider. Therefore, to help ensure successful installation of such applications, make sure that the cluster includes a running server.

Scoping side effects

A side effect of the name lookup is that if an application server is not running and a data source is defined on its server or node level with the same name as a data source at the cell level, the cell level data source takes precedence. This means that you might end up using different data sources during deployment than you use at run time.

Attention: Avoid name clashes. If you define data sources at the cell level manually, use JNDI names that have the scope of the cluster name or server name and node name appended to them, for example, `jdbc/BPEDB_cluster_name`.

Uninstalling business process and human task applications, using the administrative console

To uninstall an enterprise application that contains business processes or human tasks, perform the following actions:

1. Stop all process and task templates in the application.

   This action prevents the creation of process and task instances.

   a. Click **Applications** → **Enterprise Applications** in the administrative console navigation pane.

   b. Select the application that you want to stop.

   c. Under Related Items, click **EJB Modules**, then select an Enterprise JavaBeans (EJB) module. If you have more than one EJB module, select the EJB module that corresponds to the Service Component Architecture (SCA) module that contains the business process or human task. You can find the corresponding EJB module by appending EJB to the SCA module name. For example, if your SCA module was named `TestProcess`, the EJB module is `TestProcessEJB.jar`.

   d. Under Additional Properties, click **Business Processes** or **Human Tasks**, or both, as appropriate.

   e. Select all process and task templates by clicking the appropriate check box.

   f. Click **Stop**.

   Repeat this step for all EJB modules that contain business process templates or human task templates.

2. Verify that the database, at least one application server for each cluster, and the stand-alone server where the application is deployed are running.

   In a Network Deployment (ND) environment, the deployment manager, all ND-managed stand-alone application servers, and at least one application server must be running for each cluster where the application is installed.

3. Verify that no process instances or task instances are running nor that any are in end states with the autoDelete flag set to false.

   If necessary, an administrator can use Business Process Choreographer Explorer to delete any process or task instances.

4. Stop and uninstall the application:

   a. Click **Applications** → **Enterprise Applications** in the administrative console navigation pane.

   b. Select the application that you want to uninstall and click **Stop**.

   This step fails if any process instances or task instances still exist in the application.

   c. Select again the application that you want to uninstall, and click **Uninstall**.

   d. Click **Save** to save your changes.

The application is uninstalled.

Uninstalling business process and human task applications, using administrative commands

Administrative commands provide an alternative to the administrative console for uninstalling applications that contain business processes or human tasks.

If global security is enabled, verify that your user ID has operator authorization.

Ensure that the server process to which the administration client connects is running.

- In an ND environment, the server process is the deployment manager.
- In a stand-alone environment, the server process is the application server.

To ensure that the administrative client automatically connects to the server process, do not use the `-conntype NONE` option as a command option.

The following steps describe how to use the bpcTemplates.jacl script to uninstall applications that contain business process templates or human task templates. You must stop a template before you can uninstall the application to which it belongs. You can use the bpcTemplates.jacl script to stop and uninstall templates in one step.

Before you uninstall applications, you can delete process instances or task instances associated with the templates in the applications, for example, using Business Process Choreographer Explorer. You can also use the **-force** option with the bpcTemplates.jacl script to delete any instances associated with the templates, stop the templates, and uninstall them in one step.

CAUTION:
Because this option deletes all process instance and task instance data, you should use this option with care.

1. Change to the Business Process Choreographer samples directory. Type the following:

   ```
   cd install_root/ProcessChoreographer/admin
   ```

2. Stop the templates and uninstall the corresponding application.

   ```
   install_root/bin/wsadmin -f bpcTemplates.jacl
                               [-user user_name]
                               [-password user password]
                               -uninstall application_name
                               [-force]
   ```

   Where:

   user_name
   If global security is enabled, provide the user ID for authentication.

   user_password
   If global security is enabled, provide the user password for authentication.

   application_name
   If global security is enabled, provide the user password for authentication.

   -force
   Causes any running instances to be stopped and deleted before the application is uninstalled.

The application is uninstalled.

Chapter 5. Installing applications with embedded WebSphere Adapters

If an application is developed with a WebSphere Adapter embedded, the adapter is deployed with the application. You do not need to install the adapter separately. The steps to install an application with an embedded adapter are described.

This task should only be performed if the application is developed with an embedded WebSphere Adapter.

1. Assemble an application with resource adapter archive (RAR) modules in it. See Assembling applications.
2. Install the application following the steps in Installing a new application. In the Map modules to servers step, specify target servers or clusters for each RAR file. Be sure to map all other modules that use the resource adapters defined in the RAR modules to the same targets. Also, specify the Web servers as targets that serve as routers for requests to this application. The plug-in configuration file (plugin-cfg.xml) for each Web server is generated based on the applications that are routed through it.

   Note: When installing a RAR file onto a server, WebSphere Process Server looks for the manifest (MANIFEST.MF) for the connector module. It looks first in the connectorModule.jar file for the RAR file and loads the manifest from the connectorModule.jar file. If the class path entry is in the manifest from the connectorModule.jar file, then the RAR uses that class path. To ensure that the installed connector module finds the classes and resources that it needs, check the Class path setting for the RAR using the console. For more information, see Resource Adapter settings and WebSphere relational resource adapter settings.
3. Save the changes. Click **Finish > Save**.
4. Create connection factories for the newly installed application.
   a. Open the administrative console.
   b. Select the newly installed application Click **Applications > Enterprise Applications >** ***application name***.
   c. Click **Connector Modules** in the Related Items section of the page.
   d. Select the RAR file. Click on ***filename.rar***.
   e. Click **Resource adapter** in the Additional Properties section of the page.
   f. Click **J2C Connection Factories** in the Additional Properties section of the page.
   g. Click on an **existing connection factory** to update it, or **New** to create a new one.

      Note: If the WebSphere Adapter was configured using an EIS Import or EIS Export a ConnectionFactory or ActivationSpec will exist and can be updated.

   Linux and UNIX: If you install an adapter that includes native path elements, consider the following: If you have more than one native path element, and one of the native libraries (native library A) is dependent on another library (native

© Copyright IBM Corp. 2006, 2007

library B), then you must copy native library B to a system directory. Because of limitations on most UNIX® systems, an attempt to load a native library does not look in the current directory.

After you create and save the connection factories, you can modify the resource references defined in various modules of the application and specify the Java Naming and Directory Interface (JNDI) names of the connection factories wherever appropriate.

Note: A given native library can only be loaded one time for each instance of the Java virtual machine (JVM). Because each application has its own classloader, separate applications with embedded RAR files cannot both use the same native library. The second application receives an exception when it tries to load the library.

If any application deployed on the application server uses an embedded RAR file that includes native path elements, then you must always ensure that you shut down the application server cleanly, with no outstanding transactions. If the application server does not shut down cleanly it performs recovery upon server restart and loads any required RAR files and native libraries. On completion of recovery, do not attempt any application-related work. Shut down the server and restart it. No further recovery is attempted by the application server on this restart, and normal application processing can proceed.

WebSphere Adapter

A WebSphere Adapter (or JCA Adapter, or J2C Adapter) is a system-level software driver that a Java application uses to connect to an enterprise information system (EIS). WebSphere Adapters conform to version 1.5 of the JCA specification.

A WebSphere Adapter plugs into an application server and provides connectivity between the EIS, the application server, and the enterprise application.

An application server vendor extends its system once to support the J2EE Connector Architecture (JCA) and is then assured of seamless connectivity to multiple EISs. Likewise, an EIS vendor provides one standard WebSphere Adapter with the capability to plug into any application server that supports the connector architecture.

WebSphere Process Server provides the WebSphere Relational Resource Adapter (RRA) implementation. This WebSphere Adapter provides data access through JDBC calls to access the database dynamically. The connection management is based on the JCA connection management architecture. It provides connection pooling, transaction, and security support. WebSphere Process Server version 6.0 supports JCA version 1.5.

Data access for container-managed persistence (CMP) beans is managed by the WebSphere Persistence Manager indirectly. The JCA specification supports persistence manager delegation of the data access to the WebSphere Adapter without specific knowledge of the back-end store. For the relational database access, the persistence manager uses the relational resource adapter to access the data from the database. You can find the supported database platforms for the JDBC API at the WebSphere Process Server prerequisite Web site.

IBM® supplies resource adapters for many enterprise systems separately from the WebSphere Process Server package, including (but not limited to): the Customer

Information Control System (CICS®), Host On-Demand (HOD), Information Management System (IMS™), and Systems, Applications, and Products (SAP) R/3.

In WebSphere Process Server, EIS Imports and EIS Exports are used to interface with WebSphere Adapters. As an alternative, applications with WebSphere Adapters can be written by developing EJB session beans or services with tools such as Rational® Application Developer. The session bean uses the javax.resource.cci interfaces to communicate with an enterprise information system through the WebSphere Adapter.

WebSphere Adapter deployment considerations

The deployment of WebSphere Adapters requires specific options regarding scope.

You can deploy a WebSphere Adapter in two ways, using the administrative console:

- Standalone - the adapter is installed at the node level and is not associated with a specific application.

  Note: Deployment of standalone WebSphere Adapters is not supported in WebSphere Process Server v6.0.
- Embedded - the adapter is part of an application, deploying the application also deploys the adapter.

For embedded WebSphere Adapters:

- the RAR file can be application-scoped within an SCA module (with EIS imports or exports).
- the RAR file can be application-scoped within a non-SCA module. The application itself, containing the EIS imports and exports, is a separate SCA module.

You should not install standalone WebSphere Adapters.

Note: The administrative console does not preclude the installation of standalone WebSphere Adapters, but this should not be done. WebSphere Adapters should be embedded in applications.

Only embedded WebSphere Adapters are appropriate for deployment in WebSphere Process Server. Furthermore, deployment of an embedded WebSphere Adapter is only supported for RAR files that are application-scoped within an SCA module; deployment in a non-SCA module is not supported.

Installing Standalone WebSphere Adapters

If you intend to use a standalone WebSphere Adapter you should install it, as described here. You can alternatively use an embedded adapter, which is installed automatically as part of the installation of the associated application.

Note: WebSphere Adapters should be embedded in applications. Standalone WebSphere Adapters are not support in WebSphere Process Server v6.0. These instructions are for reference only.

You should configure the database before installing the adapter.

You must have access to, and be part of the necessary security role for, the administrative console to perform this task.

1. Open the Install RAR file dialog window. On the administrative console:
   a. Expand **Resources**
   b. Click **Resource Adapters**
   c. Select the scope at which you want to define this resource adapter. (This scope becomes the scope of your connection factory). You can choose cell, node, cluster, or server.
   d. Click **Install RAR**

   A window opens in which you can install a JCA connector and create, for it, a WebSphere Adapter. You can also use the **New** button, but the **New** button creates only a new resource adapter (the JCA connector must already be installed on the system).

   Note: When installing a RAR file using this dialog, the scope you define on the Resource Adapters page has no effect on where the RAR file is installed. You can install RAR files only at the node level. The node on which the file is installed is determined by the scope on the Install RAR page. (The scope you set on the Resource Adapters page determines the scope of the new resource adapters, which you can install at the server, node, or cell level.)
2. Install the RAR file

   From the dialog, install the WebSphere Adapter in the following manner:
   a. Browse to the location of the JCA connector. If the RAR file is on the local workstation select **Local Path** and browse to find the file. If the RAR file is on a network server, select **Server path** and specify the fully qualified path to the file.
   b. Click **Next**
   c. Enter the resource adapter name and any other properties needed under General Properties. If you install a J2C Resource Adapter that includes native path elements, consider the following: If you have more than one native path element, and one of the native libraries (native library A) is dependent on another library (native library B), then you must copy native library B to a system directory. Because of limitations on most UNIX systems, an attempt to load a native library does not look in the current directory.
   d. Click **OK**.

WebSphere Adapter applications as members of clusters

WebSphere Adapter module applications can be cloned as members of a cluster under certain conditions.

WebSphere Adapter module applications can be one of three types, depending on the flow of information through the adapter:

- A WebSphere Adapter application with only EIS exports - only inbound traffic.
- A WebSphere Adapter application with only EIS imports - only outbound traffic.
- A WebSphere Adapter application with both EIS imports and exports - bidirectional traffic.

Clusters are used to provide scalability and availability to your applications in a network deployment environment.

WebSphere Adapter module applications that have either inbound or bidirectional traffic, cannot be cloned as members of a cluster. An application with purely outbound traffic can be cloned as a member of a cluster.

An application that has an inbound or bidirectional WebSphere Adapter (that is, including EIS exports) can still be given availability in a network deployment by use of an external Operating System High Availability (HA) management software package, such as HACMP™, Veritas or Microsoft® Cluster Server.

WebSphere Business Integration Adapter applications as members of clusters

WebSphere Business Integration Adapter module applications can be cloned as members of a cluster under certain conditions.

WebSphere Business Integration Adapter module applications can be one of three types, depending on the flow of information through the adapter:

- A WebSphere Business Integration Adapter application with only EIS exports - only inbound traffic.
- A WebSphere Business Integration Adapter application with only EIS imports - only outbound traffic.
- A WebSphere Business Integration Adapter application with both EIS imports and exports - bidirectional traffic.

Clusters are used to provide scalability and availability to your applications in a network deployment environment.

WebSphere Business Integration Adapter module applications that have either inbound or bidirectional traffic, cannot be cloned as members of a cluster. An application with purely outbound traffic can be cloned as a member of a cluster.

An application which has inbound or bidirectional WebSphere Business Integration Adapter (i.e., including EIS exports) can still be given availability in a network deployment by use of an external Operating System High Availability (HA) management software package, such as HACMP, Veritas or Microsoft Cluster Server.

Chapter 6. Installing EIS applications

An EIS application module can be deployed to a J2EE platform. The deployment results in an application, packaged as an EAR file deployed to the server. All the J2EE artifacts and resources are created, the application is configured and ready to be run.

The deployment to the J2EE platform creates the following J2EE artifacts and resources:

Table 1. Mapping from bindings to J2EE artifacts

Binding in the SCA module	Generated J2EE artifacts	Created J2EE resources
EIS Import	Resource References generated on the module Session EJB.	ConnectionFactory
EIS Export	Message Driven Bean, generated or deployed, depending on the listener interface supported by the Resource Adapter.	ActivationSpec
JMS Import	Message Driven Bean (MDB) provided by the runtime is deployed, resource references generated on the module Session EJB. Note that the MDB is only created if the import has a receive destination.	• ConnectionFactory • ActivationSpec • Destinations
JMS Export	Message Driven Bean provided by the runtime is deployed, resource references generated on the module Session EJB	• ActivationSpec • ConnectionFactory • Destinations

When the import or export defines a resource like a ConnectionFactory, the resource reference is generated into the deployment descriptor of the module Stateless Session EJB. Also, the appropriate binding is generated into the EJB binding file. The name, to which resource reference is bound, is either the value of the target attribute, if one is present, or default JNDI lookup name given to the resource, based on the module name and import name.

Upon deployment, the implementation locates the module session bean and uses it to lookup the resources.

During deployment of the application to the server, the EIS installation task will check for the existence of the element resource to which it is bound. If it does not exist, and the SCDL file specifies at least one property, the resource will be created and configured by the EIS installation task. If the resource does not exist, no action is taken, it is assumed that resource will be created before execution of the application.

When the JMS Import is deployed with a receive destination, a Message Driver Bean (MDB) is deployed. It listens for replies to requests that have been sent out. The MDB is associated (listens on) the Destination sent with the request in the

© Copyright IBM Corp. 2006, 2007

JMSreplyTo header field of the JMS message. When the reply message arrives, the MDB uses its correlation ID to retrieve the callback information stored in the callback Destination and then invokes the callback object.

The installation task creates the ConnectionFactory and three destinations from the information in the import file. In addition, it creates the ActivationSpec to enable the runtime MDB to listen for replies on the receive Destination. The properties of the ActivationSpec are derived from the Destination/ConnectionFactory properties. If the JMS provider is a SIBus Resource Adapter, the SIBus Destinations corresponding to the JMS Destination are created.

When the JMS Export is deployed, a Message Driven Bean (MDB) (not the same MDB as the one deployed for JMS Import) is deployed. It listens for the incoming requests on the receive Destination and then dispatches the requests to be processed by the SCA. The installation task creates the set of resources similar to the one for JMS Import, an ActivationSpec, ConnectionFactory used for sending a reply and two Destinations. All the properties of these resources are specified in the export file. If the JMS provider is an SIBus Resource Adapter, the SIBus Destinations corresponding to JMS Destination are created.

Deploying an EIS application module to the J2SE platform

The EIS Module can be deployed to J2SE platform however only EIS Import will be supported.

You need to create an EIS application module with a JMS Import binding in the WebSphere Integration Development environment before commencing this task.

An EIS application module would be furnished with a JMS Import binding when you want to access EIS systems asynchronously through the use of message queues.

Deploying to the J2SE platform is the only instance where the binding implementation can be executed in the non-managed mode. The JMS Binding requires asynchronous and JNDI support, neither of which is provided by the base service component architecture or the J2SE. The J2EE Connector Architecture does not support non-managed inbound communication thus eliminating EIS Export.

When the EIS application module with the EIS Import is deployed to J2SE, in addition to the module dependencies, the WebSphere Adapter used by the import has to be specified as the dependency, in the manifest or any other form supported by SCA.

Deploying an EIS application module to the J2EE platform

The deployment of EIS module to the J2EE platform results in an application, packaged as an EAR file deployed to the server. All the J2EE artifacts and resources are created, the application is configured and ready to be run.

You need to create an EIS module with a JMS Import binding in the WebSphere Integration Development environment before commencing this task.

The deployment to the J2EE platform creates the following J2EE artifacts and resources:

Table 2. Mapping from bindings to J2EE artifacts

Binding in the SCA module	Generated J2EE artifacts	Created J2EE resources
EIS Import	Resource References generated on the module Session EJB.	ConnectionFactory
EIS Export	Message Driven Bean, generated or deployed, depending on the listener interface supported by the Resource Adapter.	ActivationSpec
JMS Import	Message Driven Bean (MDB) provided by the runtime is deployed, resource references generated on the module Session EJB. Note that the MDB is only created if the import has a receive destination.	• ConnectionFactory • ActivationSpec • Destinations
JMS Export	Message Driven Bean provided by the runtime is deployed, resource references generated on the module Session EJB	• ActivationSpec • ConnectionFactory • Destinations

When the import or export defines a resource like a ConnectionFactory, the resource reference is generated into the deployment descriptor of the module Stateless Session EJB. Also, the appropriate binding is generated into the EJB binding file. The name, to which resource reference is bound, is either the value of the target attribute, if one is present, or default JNDI lookup name given to the resource, based on the module name and import name.

Upon deployment, the implementation locates the module session bean and uses it to lookup the resources.

During deployment of the application to the server, the EIS installation task will check for the existence of the element resource to which it is bound. If it does not exist, and the SCDL file specifies at least one property, the resource will be created and configured by the EIS installation task. If the resource does not exist, no action is taken, it is assumed that resource will be created before execution of the application.

When the JMS Import is deployed with a receive destination, a Message Driver Bean (MDB) is deployed. It listens for replies to requests that have been sent out. The MDB is associated (listens on) the Destination sent with the request in the JMSreplyTo header field of the JMS message. When the reply message arrives, the MDB uses its correlation ID to retrieve the callback information stored in the callback Destination and then invokes the callback object.

The installation task creates the ConnectionFactory and three destinations from the information in the import file. In addition, it creates the ActivationSpec to enable the runtime MDB to listen for replies on the receive Destination. The properties of the ActivationSpec are derived from the Destination/ConnectionFactory properties. If the JMS provider is a SIBus Resource Adapter, the SIBus Destinations corresponding to the JMS Destination are created.

When the JMS Export is deployed, a Message Driven Bean (MDB) (not the same MDB as the one deployed for JMS Import) is deployed. It listens for the incoming requests on the receive Destination and then dispatches the requests to be processed by the SCA. The installation task creates the set of resources similar to the one for JMS Import, an ActivationSpec, ConnectionFactory used for sending a reply and two Destinations. All the properties of these resources are specified in the export file. If the JMS provider is an SIBus Resource Adapter, the SIBus Destinations corresponding to JMS Destination are created.

Chapter 7. Troubleshooting a failed deployment

This topic describes the steps to take to determine the cause of a problem when deploying an application. It also presents some possible solutions.

This topic assumes the following things:
- You have a basic understanding of debugging a module.
- Logging and tracing is active while the module is being deployed.

The task of troubleshooting a deployment begins after you receive notification of an error. There are various symptoms of a failed deployment that you have to inspect before taking action.

1. Determine if the application installation failed.

   Examine the SystemOut.log file for messages that specify the cause of failure. Some of the reasons an application might not install include the following:
   - You are attempting to install an application on multiple servers in the same Network Deployment cell.
   - An application has the same name as an existing module on the Network Deployment cell to which you are installing the application.
   - You are attempting to deploy J2EE modules within an EAR file to different target servers.

   Important: If the installation has failed and the application contains services, you must remove any SIBus destinations or J2C activation specifications created prior to the failure before attempting to reinstall the application. The simplest way to remove these artifacts is to click **Save > Discard all** after the failure. If you inadvertently save the changes, you must manually remove the SIBus destinations and J2C activation specifications (see Deleting SIBus destinations and Deleting J2C activation specifications in the Administering section).
2. If the application is installed correctly, examine it to determine if it started successfully.

   If the application did not start successfully, the failure occurred when the server attempted to initiate the resources for the application.
   a. Examine the SystemOut.log file for messages that will direct you on how to proceed.
   b. Determine if resources required by the application are available and/or have started successfully.

      Resources that are not started prevent an application from running. This protects against lost information. The reasons for a resource not starting include:
      - Bindings are specified incorrectly
      - Resources are not configured correctly
      - Resources are not included in the resource archive (RAR) file
      - Web resources not included in the Web services archive (WAR) file
   c. Determine if any components are missing.

      The reason for missing a component is an incorrectly built enterprise archive (EAR) file. Make sure that all of the components required by the

© Copyright IBM Corp. 2006, 2007

module are in the correct folders on the test system on which you built the Java archive (JAR) file. Refer to **Developing and deploying modules > Overview of preparing and installing modules > Preparing to deploy to a server** for additional information.

3. Examine the application to see if there is information flowing through it.

   Even a running application can fail to process information. Reasons for this are similar to those mentioned in step 2b on page 39.

   a. Determine if the application uses any services contained in another application. Make sure that the other application is installed and has started successfully.

   b. Determine if the import and export bindings for devices contained in other applications used by the failing application are configured correctly. Use the administrative console to examine and correct the bindings.

4. Correct the problem and restart the application.

Deleting J2C activation specifications

The system builds J2C application specifications when installing an application that contains services. There are occasions when you must delete these specifications before reinstalling the application.

If you are deleting the specification because of a failed application installation, make sure the module in the Java Naming and Directory Interface (JNDI) name matches the name of the module that failed to install. The second part of the JNDI name is the name of the module that implemented the destination. For example in sca/SimpleBOCrsmA/ActivationSpec, **SimpleBOCrsmA** is the module name.

Delete J2C activation specifications when you inadvertently saved a configuration after installing an application that contains services and do not require the specifications.

1. Locate the activation specification to delete.

   The specifications are contained in the resource adapter panel. Navigate to this panel by clicking **Resources > Resource adapters**.

   a. Locate the **Platform Messaging Component SPI Resource Adapter.**

      To locate this adapter, you must be at the **node** scope for a stand alone server or at the **server** scope in a Network Deployment environment.

2. Display the J2C activation specifications associated with the Platform Messaging Component SPI Resource Adapter.

   Click on the resource adapter name and the next panel displays the associated specifications.

3. Delete all of the specifications with a **JNDI Name** that matches the module name that you are deleting.

   a. Click the check box next to the appropriate specifications.

   b. Click **Delete.**

The system removes selected specifications from the display.

Save the changes.

Deleting SIBus destinations

SIBus destinations are the connections that make services available to applications. There will be times that you will have to remove destinations.

If you are deleting the destination because of a failed application installation, make sure the module in the destination name matches the name of the module that failed to install. The second part of the destination is the name of the module that implemented the destination. For example in sca/SimpleBOCrsmA/component/test/sca/cros/simple/cust/Customer, **SimpleBOCrsmA** is the module name.

Delete SIBus destinations when you inadvertently saved a configuration after installing an application that contains services or you no longer need the destinations.

Note: This task deletes the destination from the SCA system bus only. You must remove the entries from the application bus also before reinstalling an application that contains services (see Deleting J2C activation specifications in the Administering section of this information center.

1. Log into the administrative console.
2. Display the destinations on the SCA system bus.

   Navigate to the panel by clicking **Service integration > buses**
3. Select the SCA system bus destinations.

   In the display, click on **SCA.SYSTEM.*cellname*.Bus**, where *cellname* is the name of the cell that contains the module with the destinations you are deleting.
4. Delete the destinations that contain a module name that matches the module that you are removing.
   a. Click on the check box next to the pertinent destinations.
   b. Click **Delete.**

The panel displays only the remaining destinations.

Delete the J2C activation specifications related to the module that created these destinations.

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 grant 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 106-0032, 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 the 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.

© Copyright IBM Corp. 2006, 2007

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 Corporation
577 Airport Blvd., Suite 800
Burlingame, CA 94010
U.S.A.

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 document 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 measurements may have been made on development-level systems and there is no guarantee that these measurements will be the same on generally available systems. Furthermore, some measurements may have been estimated through extrapolation. Actual results may vary. Users of this document should verify the applicable data for their specific environment.

Information concerning non-IBM products was obtained from the suppliers of those products, their published announcements or other publicly available sources. IBM has not tested those products and cannot confirm the accuracy of performance, compatibility or any other claims related to non-IBM products. Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products.

All statements regarding IBM's future direction or intent are subject to change or withdrawal without notice, and represent goals and objectives only.

This information is for planning purposes only. The information herein is subject to change before the products described become available.

This information contains examples of data and reports used in daily business operations. To illustrate them as completely as possible, the examples include the 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.

COPYRIGHT LICENSE:

This information contains sample application programs in source language, which illustrate programming techniques on various operating platforms. You may copy, modify, and distribute these sample programs in any form without payment to IBM, for the purposes of developing, using, marketing or distributing application programs conforming to the application programming interface for the operating platform for which the sample programs are written. These examples have not been thoroughly tested under all conditions. IBM, therefore, cannot guarantee or imply reliability, serviceability, or function of these programs.

If you are viewing this information softcopy, the photographs and color illustrations may not appear.

Programming interface information

Programming interface information, if provided, is intended to help you create application software using this program.

General-use programming interfaces allow you to write application software that obtain the services of this program's tools.

However, this information may also contain diagnosis, modification, and tuning information. Diagnosis, modification and tuning information is provided to help you debug your application software.

Warning: Do not use this diagnosis, modification, and tuning information as a programming interface because it is subject to change.

Trademarks and service marks

The following terms are trademarks or registered trademarks of International Business Machines Corporation in the United States or other countries, or both: IBM, IBM (logo), AIX, CICS, Cloudscape, DB2, DB2 Connect, DB2 Universal Database, developerWorks, Domino, IMS, Informix, iSeries, Lotus, MQSeries, MVS, OS/390, Passport Advantage, pSeries, Rational, Redbooks, Tivoli, WebSphere, z/OS, zSeries

Java and all Java-based trademarks are trademarks of Sun Microsystems, Inc. in the United States, other countries, or both.

Microsoft and Windows are trademarks of Microsoft Corporation in the United States, other countries, or both.

UNIX is a registered trademark of The Open Group in the United States and other countries.

Linux is a trademark of Linus Torvalds in the United States, other countries, or both.

Other company, product, or service names may be trademarks or service marks of others.

This product includes software developed by the Eclipse Project (http://www.eclipse.org/).

IBM WebSphere Process Server for z/OS version 6.0.2

Printed in USA