IMS

IMS Connector for Java 9.1.0.1 and 9.1.0.2 Online Documentation for Rational Application Developer 6.0

SC18-9593-01

IMS

IMS Connector for Java 9.1.0.1 and 9.1.0.2 Online Documentation for Rational Application Developer 6.0

SC18-9593-01

Note
Before using this information and the product it supports, read the information in Notices at the end of this book.

Second Edition (September 2005)

Contents

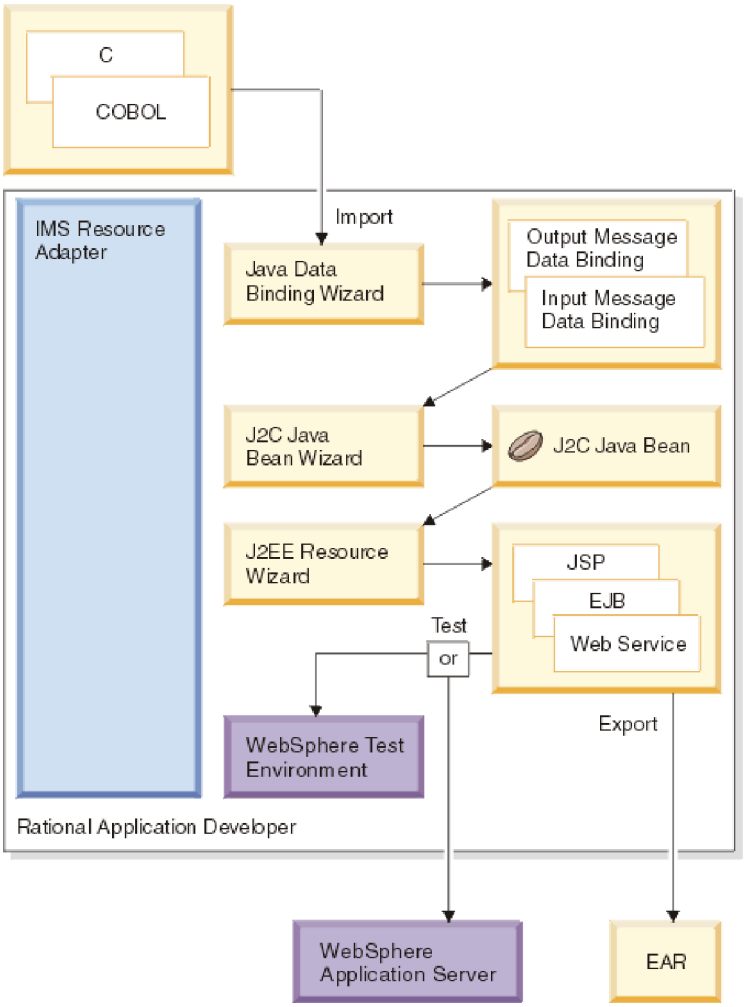
Chapter 1. What is the IMS resource adapter?

The IMS resource adapter is used by Java applications to access IMS transactions running on host IMS systems. The IMS resource adapter is available in a number of Java integrated development environments (IDEs) provided by IBM. Two of these IDEs are WebSphere Studio Application Developer Integration Edition and Rational Application Developer with the optional J2EE Connector Architecture (J2C) feature. The IMS resource adapter is also used at runtime by WebSphere Application Server when a Java application accesses an IMS transaction running on a host IMS system. The IMS resource adapter is also called IMS Connector for Java.

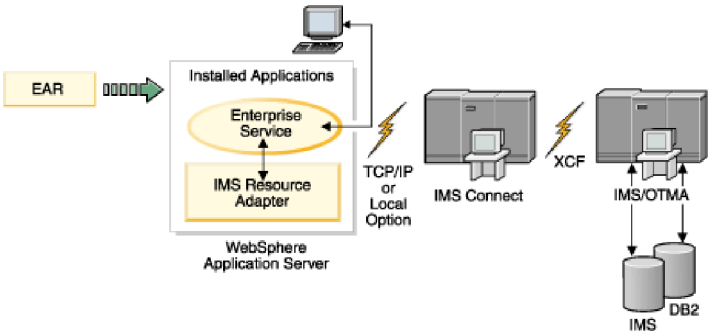
The process of using Rational Application Developer with the J2C Connector tools to build a Java application that runs an IMS transaction is summarized by the following steps:

1. Import C or COBOL definitions of the IMS transaction input and output messages into the Java Data Binding wizard to map to Java data structures. This wizard creates Java data bindings for the input and output messages.
2. Provide Java data bindings to the J2C Java Bean wizard. This wizard creates a J2C Java bean with methods that can be used to run IMS transactions on the host.
3. Provide the J2C Java bean to the J2C dynamic wizard used to create a J2EE resource. This J2EE resource can be deployed to WebSphere Application Server and used to run your IMS transactions. The types of J2EE resources that can be created from a J2C Java bean are:
   - JSP
   - Web Service
   - EJB
4. Test the J2EE resource directly from the development environment using the WebSphere test environment provided with Rational Application Developer.
5. Export the J2EE resource, packaged as an EAR file by Rational Application Developer, so that it can be deployed to and run on a stand alone WebSphere Application Server.

The following figure illustrates the use of the IMS resource adapter during development:



At run time, the IMS resource adapter is used with IBM WebSphere® Application Server. When a Java™ application runs, it submits a transaction request to IMS™ through the host product, IMS Connect. The IMS resource adapter communicates with IMS Connect using TCP/IP or Local Option. IMS Connect then sends the transaction request to IMS OTMA using XCF (Cross-system Coupling Facility), and the transaction runs in IMS. The response is returned to the Java application using the same path. The following figure illustrates the run-time process:



Two IMS resource adapters are provided as part of the J2C feature of Rational Application Developer. IMS Connector for Java Version 9.1.0.1.*x* is based on Version 1.0 of the J2EE Connector Architecture and IMS Connector for Java Version 9.1.0.2.*x* is based on Version 1.5 of the J2EE Connector Architecture. A license for IMS V9.1 is required to run an application that uses IMS Connector for Java V9.1.0.*x*.*x*. This

information does not describe the J2EE Connector architecture in general. For information on the JCA architecture and its concepts, see the J2EE Connector Architecture Specification at http://java.sun.com/j2ee/download.html.

Both IMS resource adapter:

- Provides global transaction and two-phase-commit support
- Provides run as thread identity support
- Supports component-managed and container-managed security
- Supports pooling and reuse of connections
- Supports SSL communication between IMS Connector for Java and IMS Connect
- Supports both commit mode 1 and commit mode 0 IMS transactions
- Supports the retrieval of output messages queued as the result of a failed commit mode 0 interaction or by insertion to an alternate PCB
- Supports conversational processing
- Provides control of whether undelivered output for commit mode 0 interactions on shareable persistent socket connections is queued or discarded. This function is controlled by the **purgeAsyncOutput** property.
- Supports specification of the name of a destination for undelivered output for commit mode 0 interactions on shareable persistent socket connections. This function is controlled by the **reRoute** flag and **reRouteName** properties.
- Provides enhanced control of the retrieval of undelivered output with the introduction of two new interaction verbs: SYNC_RECEIVE_ASYNCOUTPUT_SINGLE_WAIT and SYNC_RECEIVE_ASYNCOUTPUT_SINGLE_NOWAIT.
- Supports the use of RACF keyrings as SSL keystores and truststores.

See IMS resource adapter APIs for additional information on the IMS resource adapter's J2C classes and interfaces.

The IMS resource adapter is included in WebSphere Studio Application Developer Integration Edition and the optional J2EE Connector Architecture (J2C) feature of Rational Application Developer for use in the development of Java applications. The IMS resource adapter runtime is a component of the IMS product. It is packaged as a Resource Adapter Archive (RAR) file and can be deployed to WebSphere Application Server for use by J2EE applications. The IMS resource adapter runtime is available for download from the IMS Web site (www.ibm.com/software/data/ims) and is also available for installation on z/OS using SMP/E.

The IMS resource adapter is primarily intended for use by services that submit transactions to IMS. However, the IMS resource adapter can also be used by services that submit IMS commands to IMS.

Chapter 2. Preparing to use the IMS resource adapter

If you are planning to develop a Java application that runs an IMS transaction, you must import the IMS resource adapter into your workbench.

There are two IMS resource adapters provided as part of the J2C feature of Rational Application Developer. IMS Connector for Java Version 9.1.0.1.1X is based on Version 1.0 of the J2EE Connector Architecture and IMS Connector for Java Version 9.1.0.2.*x* is based on Version 1.5 of the J2EE Connector Architecture.

The version of the IMS resource adapter that you use depends on which J2EE Connector Architecture version your application uses.

The J2C wizard in Rational Application Developer enables you to create J2C applications, either as a standalone program, or as added functionality to existing applications. This wizard also dynamically imports your selected resource adapter.

For information on how to deploy the IMS resource adapter on WebSphere® Application Server, see the file Howto.html that is packaged with the runtime component of IMS Connector for Java.

Prerequisites for using the IMS resource adapter

This topic describes the prerequisites for using the IMS resource adapter as well as the supported software configurations.

This topic describes the prerequisites for using the IMS resource adapter as well as the supported software configurations.

Rational Application Developer with the optional J2EE Connector Architecture (J2C) feature includes two versions of the IMS resource adapter:

- IMS Connector for Java Version 9.1.0.2.*x*

  This version of the IMS resource adapter is based on Version 1.5 of the J2EE Connector Architecture (JCA 1.5). IMS Connector for Java Version 9.1.0.2.*x* is not included in WebSphere Studio Application Developer Integration Edition. Because IMS Connector for Java Version 9.1.0.2.*x* is a JCA 1.5 resource adapter, it will only run in a JCA 1.5 application server or WebSphere Application Server Version 6.0 (or above) for distributed and z/OS platforms.

  In the future, enhancements to IMS Connector for Java will be made to the JCA 1.5 version only.

- IMS Connector for Java Version 9.1.0.1.*x*

  This version of the IMS resource adapter is based on Version 1.0 of the J2EE Connector Architecture (JCA 1.0). This version of IMS Connector for Java runs with WebSphere Application Server Version 5.0.2 and above for distributed and z/OS platforms.

  WebSphere Studio Application Developer Integration Edition Version 5.1.1 also contains two versions of the IMS resource adapter:

- IMS Connector for Java Version 2.2.*x*

  This version of the IMS resource adapter is based on Version 1.0 of the J2EE Connector Architecture (JCA 1.0). This version of IMS Connector for Java runs with WebSphere Application Server Version 5.0.2 and above for distributed and z/OS platforms.

- IMS Connector for Java Version 9.1.0.1.*x*

  This version of the IMS resource adapter is based on Version 1.0 of the J2EE Connector Architecture (JCA 1.0), runs with WebSphere Application Server Version 5.0.2 and above for distributed and z/OS

platforms. IMS Connector for JavaVersion 9.1.0.1.1 is functionally equivalent to IMS Connector for Java Version 2.2.3; IMS Connector for JavaVersion 9.1.0.1.2 is functionally equivalent to IMS Connector for Java Version 2.2.4.

The following new function is included in Versions 2.2.3, 9.1.0.1.1, and 9.1.0.2 of the IMS resource adapter:

- The control of whether undelivered output for commit mode 0 interactions on shareable persistent socket connections is queued or discarded. This function is controlled by the **purgeAsyncOutput** property.
- The option to provide the name of a destination for undelivered output for commit mode 0 interactions on shareable persistent socket connections. This function is controlled by the **reRoute** flag and **reRouteName** properties.
- Enhanced control of the retrieval of undelivered output with the introduction of two new interaction verbs: SYNC_RECEIVE_ASYNCOUTPUT_SINGLE_WAIT and SYNC_RECEIVE_ASYNCOUTPUT_SINGLE_NOWAIT.
- Support for use of RACF keyrings as SSL keystores and truststores.

In addition, IMS Connector for Java Version 9.1.0.2 adds the following JCA 1.5 performance enhancements:

- Lazy Transaction Enlistment Optimization
- Lazy Connection Association Optimization

Note: IMS Connector for Java Version 9.1.0.2 does not provide the following JCA 1.5 implementations:

- Inbound messaging, transaction, and EJB invocation processing
- Implementation of the Work Management contract
- Kerberos support

IMS Connector for Java supported configurations

IC4J 2.2.*x* supports:

- IMS Connect V2.2 plus IMS V8 and APARs or IMS V9 and APARs

IC4J 9.1.0.1.*x* and 9.1.0.2.*x* supports:

- IMS Connect V2.2 plus IMS V8 and APARs or IMS V9 and APARs
- IMS Connect V9.1 plus IMS V9 and APARs

For APAR and IFix numbers, please go to the IMS Web site at www.ibm.com/software/data/ims and link to the IMS Connector for Java web page.

Platform configurations and communication protocol considerations

The communication protocol you use depends on the platform configuration of WebSphere Application Server and IMS. The IMS resource adapter can be deployed to WebSphere Application Server for distributed platforms (AIX®, HP_UX, Linux, Linux for z/OS®, Solaris, or Windows®) and to WebSphere Application Server for z/OS. The IMS resource adapter, deployed in WebSphere Application Server, can communicate with IMS Connect using either the TCP/IP or Local Option communication protocol. Where TCP/IP uses sockets, Local Option provides non-socket access (an MVS™ program call) to IMS Connect from WebSphere Application Server for z/OS.

- If WebSphere Application Server is running on a distributed platform, you must use TCP/IP to connect to IMS Connect.
  - If you use global transaction (two-phase-commit) support with TCP/IP, RRS is required. Also, IMS Connect, IMS, and RRS must reside in the same MVS image.
- If WebSphere Application Server is running on z/OS, you can use either TCP/IP or Local Option to connect to IMS Connect depending on your configuration. For example:

- If WebSphere Application Server and IMS Connect are on the same MVS image, you can use Local Option or TCP/IP; however Local Option is recommended.
- If WebSphere Application Server and IMS Connect are on different MVS images, you must use TCP/IP.
- If you want to use global transaction support and your IMS and WebSphere Application Server are on the same MVS image, the Local Option communication protocol is recommended. If you are using global transaction support with Local Option protocol, RRS, IMS, IMS Connect, and WebSphere Application Server must be in the same MVS image.
- If you want to use global transaction support and your IMS and WebSphere Application Server are on different MVS images, you must use TCP/IP as your communication protocol. If you are using global transaction support with TCP/IP protocol, RRS, IMS, and IMS Connect must reside in the same MVS image.

The following table describes the relationship between the different platform configurations, communication protocols, and global transaction support:

Platform of WebSphere Application Server with IMS resource adapter	**Supported communication protocol**	**Global transaction (two-phase-commit) support**
AIX	TCP/IP	Yes*
HP_UX	TCP/IP	Yes*
Linux	TCP/IP	Yes*
Linux for zSeries® and S/390®	TCP/IP	Yes*
Solaris	TCP/IP	Yes*
Windows	TCP/IP	Yes*
z/OS, OS/390®	TCP/IP Local Option	Yes* Yes

* Global transaction support with TCP/IP requires IMS Connect 2.1 or later.

Chapter 3. Developing your application

This topic describes how to develop your application. There are a number of ways to develop a Java application that accesses a host IMS transaction.

One way to develop a Java application is to write the application yourself, using the J2C Common Client Interface (CCI) provided by the IMS resource adapter. Another way is to use one of IBM's integrated development environments to generate the Java application for you. Examples of these integrated development environments are WebSphere Studio Application Developer Integration Edition and Rational Application Developer with the J2EE Connector tools. The code generated by these development environments also uses the J2C Common Client Interface. Using a development environment has the following benefits:

- Wizards and guides to lead you step-by-step through the development process and generate wrappers for most code artifacts.
- Easily develop applications that are compliant with J2EE standards.
- Tools to automate mapping data sources to EJBs, to manage EJB deployment descriptors and EAR file packaging and deployment, and a built-in test client for EJBs.

If you choose to write the application yourself, you can still take advantage of the functionality of Rational Application Developer by using the wizards provided by the J2C Connector tools to build data structures for the transaction input and output messages required by your ″CCI application″.

The process of building a Java application using the J2C Connector tools of Rational Application Developer can be divided into three steps:

1. Create IMS data bindings for the input and output message of the IMS transaction.
2. Create a J2C Java bean that has a method that communicates with IMS to run the IMS transaction using the J2EE Connector Architecture. This method uses IMS data bindings created earlier.
3. Create a J2EE resource that wraps the functionality of the J2C Java bean and creates an application that can be used to run the transaction in IMS.

This process is described in more detail in the sections that follow.

Creating IMS Java data bindings

The J2C dynamic wizard in Rational Application Developer allows you to create specialized Java classes representing the input and output messages of an IMS transaction from the corresponding COBOL or C data structures of the IMS application program. These specialized Java classes are called data bindings.

Data bindings provide a Java application with methods for populating the input message with data and for retrieving data from the output message. In addition, the data bindings perform platform-related functions such as conversion between the Java (UNICODE) and host (EBCDIC) representations of the data. To create IMS Java data bindings, complete the following steps:

Note: Ensure that you are in the J2EE perspective.

1. To start the J2C dynamic wizard, from the menu bar, select **File > New > Other > J2C**.
2. Expand **J2C**.
3. Select **CICS/IMS Java Data Binding** and click **Next**.
4. On the Data Import page, you need to specify the data import configuration properties.
   a. In the Choose mapping drop down list, select **COBOL to Java** mapping.

b. Click **Browse** to select the COBOL file for which you are creating data bindings. For example, browse to *rad_install_dir* **/rad/eclipse/plugins/com.ibm.j2c.cheatsheet.content.6.0.0/samples/IMS/Phonebook/Ex01.cbl**.

c. Click **Next**.

5. In the **Importer** wizard, complete the following steps:

a. In the Platform drop down list, select the platform on which your IMS transactions will run. For example, **z/OS**.

b. In the Code page drop down list, select a different value if the data of your IMS transaction is in a codepage other than US English (IBM - 037).

c. Click **Show Advanced** to see the advanced properties. If you choose the z/OS platform, the values for all the fields are automatically filled.

d. Change the value of TRUNC from STD to **BIN**. Because most IMS programs are compiled with the TRUNC(BIN) option, it is recommended that you change the value of TRUNC from STD to BIN.

e. For this example, accept all the other default values listed in the following table.

Table 1.

Options	**Value**
Platform	z/OS
Codepage	IBM-037
Floating point format	IBM 390 Hexadecimal
External decimal sign	EBCDIC
Endian	Big
Remote integer endian	Big
Quote	DOUBLE
Trunc	BIN
Nsymbol	DBCS

f. Click **Query** to select the data structure for which you are creating a data binding. The available data structures in the COBOL file that you specified previously are displayed in the Data structures pane.

g. Select **INPUT-MSG**, the COBOL data structure used by the IMS application program to describe the input message for the IMS transaction.

h. Click **Next**.

6. On the Saving Properties page of the Import wizard, complete the following steps:

a. For the **Generation Style**, use **Default**.

b. For the **Java Project Name**, click **New**, select **Java project** as the project type, and then click **Next**.

c. In the Create a Java project page, type `PhoneBookBindings` as the Java project name, accept all other defaults, and then click **Finish**.

d. In the Saving Properties page, for the **Java Package Name**, click **New** and create a new Java package named, `sample.ims`, in the project, *PhoneBookBindings*. Then, click **Finish**.

7. Click **Finish** to save the Import properties.

8. In the Project Explorer view, expand **Other Projects > PhoneBookBindings > sample.ims**.

9. Right-click *sample.ims* package and select **New > Other > J2C > CICS/IMS Java Data Binding**, then **Next**.

10. Repeat steps 4 through 6 to create a data binding for the output message using **OUTPUT-MSG** as the data structure.

You now have a project, *PhoneBookBindings*, containing the data bindings for the input and output messages of your IMS transaction. These data bindings can now be used in one or more J2C Java beans or by a Java application that directly uses the CCI.

Creating a J2C Java bean

After you create IMS Java data bindings, you need to create a Java bean that communicates with IMS through the J2EE Connector Architecture.

This Java bean includes a method that submits a request to IMS to run the IMS transaction. This method uses the Java data bindings to build the input and output messages for the transaction. A J2C Java bean may include more than one method that runs an IMS transaction, as well as multiple data bindings for different input and output messages. The code that is generated for the J2C Java bean uses the CCI provided by the IMS resource adapter to communicate with IMS.

To create a J2C Java bean that runs an IMS transaction, complete the following steps:

1. To start the J2C dynamic wizard, from the menu bar, select **File > New > Other > J2C**.
2. Expand the J2C folder, select **J2C Java bean**, and click **Next**.
3. In the Resource Adapters Selection page, select the version of the IMS resource adapter that you wish to use. You may select either the JCA 1.0 IMS resource adapter, **IMS Connector for Java (IBM: 9.1.0.1.1)** or the JCA 1.5 IMS resource adapter, **IMS Connector for Java (IBM : 9.1.0.2)**.
4. Click **Next**.
5. In the Connection Properties page, select the **Managed** checkbox.

   Note: There are two options for the way in which connections are created between the IMS resource adapter, as used by your Java bean, and IMS Connect. This example is not a two-tiered application, so only a managed connection is applicable.

   - Managed connections are created by a construct of the J2EE Connector Architecture called a connection factory and are managed by the application server. Your Java bean accesses a connection factory using JNDI (Java Naming and Directory Interface). Managed connections are recommended. The IMS resource adapter and the application server's connection manager work together to efficiently manage connections by providing connection pooling, reuse, and persistence.
   - Non-managed connections are obtained directly through the IMS resource adapter, without collaboration with the application server. Non-managed connections are typically used by two-tiered applications, and are not pooled or reused. In addition, non-managed socket connections between the IMS resource adapter and IMS Connect are not persistent, incurring the additional overhead of opening and closing the socket for each use by an application.
6. On the Connection Properties page, next to the JNDI lookup name field, click **New**. This defines a new server instance. To define a new server instance, the resource adapter you selected in Step 3, is deployed to the server instance. To create a new server instance in your workspace, complete the following steps:
   a. In the JNDI Lookup wizard, on the Server instance selection page, select **New**.
   b. In the Define a New Server page, select the type of server you wish to create; for example, **WebSphere 6.0 Server**. Then, click **Next**.
   c. In the WebSphere Server Settings page, accept the defaults.
   d. Click **Finish**. The resource adapter you selected in Step 3 is deployed to your server instance.
   e. On the Server instance selection page, click **Next**.
7. Create and configure a J2C connection factory for the server instance you just created. The J2C Connection Factories wizard allows you to select a J2C connection factory from those that have been defined for the resource adapter you selected in Step 3. You can also provide the JNDI lookup name

of a connection factory that does not yet exist and define it later. To create and configure a J2C connection factory for your server instance, complete the following steps:

a. In the J2C Connection Factory page, enter a JNDI name for your new connection factory. For example, `imsCFac`.

b. Configure your connection factory. For TCP/IP connections to IMS Connect, at minimum, provide values for the following fields:

- In the **Host name:** field, enter the TCP/IP hostname of the IMS Connect that your application will use. For example: `MYHOST.MYCOMPANY.COM`
- In the **Port number:** field, enter the port number. For example, `9999`.
- In the **Data store name:** field, enter the target IMS datastore. For example, `IMSA`.

c. Click **Finish**. The server instance is started and initialized.

8. After you have configured your server instance and your server has started, the JNDI name of the connection factory appears in the JNDI Lookup name field. Click **Next**.
9. In the J2C Java Bean Output Properties page, complete the following steps:
   a. For the Java Project Name , click **New**.
   b. Select **Java project** as the project type, and click **Next**.
   c. In the Create a Java project page, type `PhoneBookJ2CBean` for the new Java project name, accept all other defaults, and then click **Finish**.
   d. Next to Java Package Name, click **New**.
   e. In the Java Package page, type `sample.ims` for the new Java package name and click **Finish**.
   f. For the Interface Name, specify `PB`.
   g. For the Binding Name, accept the default of `PBImpl`.
   h. Click **Next**.
10. To create a method that runs the transaction, use the Java Methods wizard and complete the following steps:
    a. Click **Add** to add a Java method to your J2C Java bean.
    b. In the Add Java Method page, type `runPB` for the Java method name, the click **Next**.
    c. On the Java Method page, click **Browse** next to the Input type field.
    d. In the Select a data type window, prime the entry field with an asterisk (*) to view the available data types.
    e. In the Matching types field, select **INPUTMSG** and click **OK** to use the INPUTMSG data binding for the method, *runPB*.
    f. Next to the Output type field, click **Browse**.
    g. In the Select a data type window, prime the entry field with an asterisk (*) to view the available data types.
    h. In the Matching types field, select **OUTPUTMSG** and then click **OK** to use the OUTPUTMSG data binding for the method, *runPB*.
    i. Click **Finish**.
11. The Java Methods wizard displays the new method, **runPB (INPUTMSG : OUTPUTMSG)** in the list of methods for the Java bean. Ensure that this method is selected.
12. In the InteractionSpec properties for 'runPB', specify the IMSInteractionSpec values. For this example, accept all defaults, then click **Finish**.

You now have a J2C Java bean in project, PhoneBookJ2CBean, that you can deploy in one or more J2EE applications using different J2EE resources.

Verifying your server instance configuration

You can verify the proper configuration of your server instance to ensure that it is running with the correct connection properties.

To verify that you have properly configured the server instance you created, complete the following steps:

1. Ensure that the server is started. In the Servers view, right-click the server instance and select **Run administrative console.** The Welcome page is displayed.
2. Logon to the administrative console.
3. In the left-hand navigation, select **Resources > Resource Adapters**.
4. On the Resource adapters page, the IMS resource adapter that you selected is displayed.
5. Select the IMS resource adapter, then in the right-hand column under Additional Properties, select **J2C Connection Factories**. You should see the J2C connection factory that you created, *imsCFac*.
6. Select **imsCFac**, then in the right-hand column under Additional Properties, select **Custom properties**. The property values that you provided for the connection factory is displayed.

Exposing InteractionSpec and ConnectionSpec properties for input as data

You can expose the properties of IMSInteractionSpec and IMSConnectionSpec for input as data so that your Java application can set or get the property values. For example, you may want to expose the userName and password properties of IMSConnectionSpec if your Java application is using component-managed EIS sign-on. Or, you may want to expose the clientID property of IMSConnectionSpec if your Java application is executing an interaction on a dedicated persistent socket connection.

To expose the properties of IMSInteractionSpec and IMSConnectionSpec for input, you must modify the interface and implementation files of your J2C Java bean before using it in an application. Typically, you expose only the properties that your Java application needs as input. The steps in this topic illustrate how to expose all the properties of IMSInteractionSpec and IMSConnectionSpec using the J2C Java bean in the project *PhoneBookJ2CBean* that was created in the topic, “Creating a J2C Java bean” on page 11.

To expose all the properties of IMSInteractionSpec and IMSConnectionSpec for input, complete the following steps:

1. Expand the project, **PhoneBookJ2CBean**, and open the interface file **PB.java** in the Java editor.
2. In the PB.java file, update the method **runPB()**. Add the arguments for the input properties of IMSInteractionSpec and IMSConnectionSpec. These arguments are used to provide input values for the exposed properties, in the same way the argument *INPUTMSGarg* is used to provide values for the input message of the IMS transaction. After you add the arguments in the method *runPB()*, the code looks like the following:

```
package sample.ims;


/**
 * @generated
 */
public interface PB {

    /**
     * @generated
     */
    public OUTPUTMSG runPB(INPUTMSG arg,
            int myCommitMode,
            int myExecutionTimeout,
            int myImsRequestType,
```

```
            int myInteractionVerb,
            String myLtermName,
            String myMapName,
            boolean myPurgeAsyncOutput,
            boolean myReRoute,
            String myReRouteName,
            int mySocketTimeout,
            String myUserName,
            String myPassword,
            String myGroupName,
            String myClientID) throws javax.resource.ResourceException;
}
```

3. Save and close the file.
4. Expand the project **PhoneBookJ2CBean** and open the binding file, **PBImpl.java** in the Java editor.
5. In the PBImpl.java file, update the javadoc for method *runPB()* by adding doclet tags for each of the properties that you wish to expose.
6. Update the signature of method *runPB()*. Add the arguments for the implementation. The arguments added to the method are referenced by the corresponding doclet tags. For information about editing J2C doclet tags, see Editing the J2C Java bean. After you add the doclet tags and update the signature of the method, the code looks like the following:

```
/**
     * @j2c.interactionSpec class="com.ibm.connector2.ims.ico.IMSInteractionSpec"
     * @j2c.interactionSpec-property name="commitMode" argumentBinding="myCommitMode"
     * @j2c.interactionSpec-property name="executionTimeout" argumentBinding="myExecutionTimeout"
     * @j2c.interactionSpec-property name="imsRequestType" argumentBinding="myImsRequestType"
     * @j2c.interactionSpec-property name="interactionVerb" argumentBinding="myInteractionVerb"
     * @j2c.interactionSpec-property name="ltermName" argumentBinding="myLtermName"
     * @j2c.interactionSpec-property name="mapName" argumentBinding="myMapName"
     * @j2c.interactionSpec-property name="purgeAsyncOutput" argumentBinding="myPurgeAsyncOutput"
     * @j2c.interactionSpec-property name="reRoute" argumentBinding="myReRoute"
     * @j2c.interactionSpec-property name="reRouteName" argumentBinding="myReRouteName"
     * @j2c.interactionSpec-property name="socketTimeout" argumentBinding="mySocketTimeout"
     *
     * @j2c.connectionSpec class="com.ibm.connector2.ims.ico.IMSConnectionSpec"
     * @j2c.connectionSpec-property name="userName" argumentBinding="myUserName"
     * @j2c.connectionSpec-property name="password" argumentBinding="myPassword"
     * @j2c.connectionSpec-property name="groupName" argumentBinding="myGroupName"
     * @j2c.connectionSpec-property name="clientID" argumentBinding="myClientID"
     *
     * @generated
     */
    public OUTPUTMSG runPB(INPUTMSG arg,
            int myCommitMode,
            int myExecutionTimeout,
            int myImsRequestType,
            int myInteractionVerb,
            String myLtermName,
            String myMapName,
            boolean myPurgeAsyncOutput,
            boolean myReRoute,
            String myReRouteName,
            int mySocketTimeout,
            String myUserName,
            String myPassword,
            String myGroupName,
            String myClientID) throws javax.resource.ResourceException {
```

7. Save and close the file. The new implementation code is generated for method *runPB()*.

For each exposed property, the doclet tag uses the *argumentBinding* attribute instead of the *value* attribute. The value of the *argumentBinding* attribute represents the method argument of the corresponding IMSInteractionSpec or IMSConnectionSpec property. You have now exposed all the IMSConnectionSpec properties for input.

Exposing InteractionSpec output properties as data

You can expose IMSInteractionSpec properties for output. Currently, the only output properties that can be exposed are asyncOutputAvailable, convEnded, and mapName. To expose these properties of IMSInteractionSpec for output, you must create a new output class and modify the interface and implementation files of your J2C Java bean before using it in an application.

Typically, you expose only the properties that your Java application needs as output. The steps in this topic illustrate how to expose all the properties of IMSInteractionSpec using the J2C Java bean in the project *PhoneBookJ2CBean* that was created in the topic, "Creating a J2C Java bean" on page 11.

To expose all the properties of IMSInteractionSpec for output, complete the following steps:

1. Expand the project **PhoneBookJ2CBean** and open the interface file, **PB.java** in the Java editor.
2. In the PB.java file, update the signature of **runPB()**. Add the arguments for the output properties of IMSInteractionSpec. These arguments are used to provide output values for the exposed properties, in the same way the argument *OUTPUTMSGarg* is used to provide values for the output message of the IMS transaction. After you add the arguments in the method *runPB()*, the code looks like the following:

```
package sample.ims;


/**
 * @generated
 */
public interface PB {

 /**
  * @generated
  */
 public sample.ims.WrapperBean runPB(sample.ims.INPUTMSG arg,
            int myCommitMode,
            int myExecutionTimeout,
            int myImsRequestType,
            int myInteractionVerb,
            String myLtermName,
            String myMapName,
            boolean myPurgeAsyncOutput,
            boolean myReRoute,
            String myReRouteName,
            int mySocketTimeout,
            String myUserName,
            String myPassword,
            String myGroupName,
            String myClientID
   ) throws javax.resource.ResourceException;
}
```

3. Create a new class, *WrapperBean*, by completing the following steps:
   a. Expand the project, **PhoneBookBindings**, right-click the **sample.ims** package, and select **New > Class**.
   b. For the name of the class, type `WrapperBean`.
   c. For the methods to create, select **Inherited abstract methods** and **Constructors from super class**.
   d. Click **Finish**.
   e. Open the *WrapperBean* class in an editor and add an import statement for **java.io.Serializable**.
   f. Modify the *WrapperBean* class so that it implements Serializable. For example:

      ```
      public class WrapperBean implements Serializable {
      ```

   g. In the *WrapperBean* class, add a private variable for the IMS Java Data binding of the output message of the IMS transaction. For example:

```
private OUTPUTMSG output;
```

h. In the *WrapperBean* class, add private variables for the properties of IMSInteractionSpec that you wish to expose: For example:

```
private boolean convEnded;
private boolean asyncOutputAvailable;
private String mapName;
```

i. Then, add get and set methods for the output message and each of the exposed properties. For example:

```
public OUTPUTMSG getOutput(){
        return output;
    }

    public boolean getConvEnded(){
        return convEnded;
    }

    public boolean getAsyncOutputAvailable(){
        return asyncOutputAvailable;
    }

    public String getMapName(){
        return mapName;
    }

    public void setOutput(OUTPUTMSG output){
        this.output = output;
    }

    public void setAsyncOutputAvailable(boolean asyncOutputAvailable){
        this.asyncOutputAvailable = asyncOutputAvailable;
    }

    public void setConvEnded(boolean convEnded){
        this.convEnded = convEnded;
    }

    public void setMapName(String mapName){
        this.mapName = mapName;
    }
```

j. Save and close the *WrapperBean* class.

4. Modify the interface file to use the new output class, *WrapperBean* by expanding **PhoneBookJ2CBean > sample.ims** and open the interface file, **PB.java**, in the Java editor.
5. Change the output of the method **runPB()**, which runs the IMS transaction, to return *WrapperBean* instead of **OUTPUTMSG**. For example:

```
public sample.ims.WrapperBean runBP(INPUTMSG arg) throws javax.resource.ResourceException;
```

6. Modify the implementation file to use the new output class, *WrapperBean* by expanding **PhoneBookJ2CBean > sample.ims** and opening the implementation file, **PBImpl.java** in the Java editor.
7. Change the output method **runPB()**, which runs the IMS transaction, to return *WrapperBean* instead of **OUTPUTMSG**. For example:

```
public sample.ims.WrapperBean runBP(INPUTMSG arg) throws javax.resource.ResourceException {
```

8. Update the javadoc for the **runPB()** method by adding doclet tags for the output properties you wish to expose. For example, the following javadoce for **runPB()** shows tags for both input and output properties:

```
/**
     * @j2c.interactionSpec class="com.ibm.connector2.ims.ico.IMSInteractionSpec"
     * @j2c.interactionSpec-property name="commitMode" argumentBinding="myCommitMode"
     * @j2c.interactionSpec-property name="executionTimeout" argumentBinding="myExecutionTimeout"
     * @j2c.interactionSpec-property name="imsRequestType" argumentBinding="myImsRequestType"
```

```
* @j2c.interactionSpec-property name="interactionVerb" argumentBinding="myInteractionVerb"
* @j2c.interactionSpec-property name="ltermName" argumentBinding="myLtermName"
* @j2c.interactionSpec-property name="mapName" argumentBinding="myMapName"
* @j2c.interactionSpec-property name="purgeAsyncOutput" argumentBinding="myPurgeAsyncOutput"
* @j2c.interactionSpec-property name="reRoute" argumentBinding="myReRoute"
* @j2c.interactionSpec-property name="reRouteName" argumentBinding="myReRouteName"
* @j2c.interactionSpec-property name="socketTimeout" argumentBinding="mySocketTimeout"
* @j2c.interactionSpec-returnProperty
*   name="convEnded"
*   outputBinding="convEnded"
* @j2c.interactionSpec-returnProperty
*   name="asyncOutputAvailable"
*   outputBinding="asyncOutputAvailable"
* @j2c.interactionSpec-returnProperty
*   name="mapName"
*   outputBinding="mapName"
*
* @j2c.connectionSpec class="com.ibm.connector2.ims.ico.IMSConnectionSpec"
* @j2c.connectionSpec-property name="userName" argumentBinding="myUserName"
* @j2c.connectionSpec-property name="password" argumentBinding="myPassword"
* @j2c.connectionSpec-property name="groupName" argumentBinding="myGroupName"
* @j2c.connectionSpec-property name="clientID" argumentBinding="myClientID"
*
* @generated
*/
```

9. Save and close the file. New implementation code is generated for method *runPB()*.

You have exposed the IMSInteractionSpec properties for output.

Creating a web page, web service, or EJB from a J2C Java bean

The final process in creating a Java application that accesses an IMS transaction is to wrap the J2C Java bean in a web page, web service, or EJB so that it can run on a J2EE application server such as WebSphere Application Server.

This example illustrates how to wrap the J2C Java bean using a JSP dynamic web application. To create a JSP dynamic web application from a J2C Java bean, complete the following steps:

1. To start the J2C dynamic wizard, from the menu bar, select **File > New > Other > J2C**.
2. With the J2C folder expanded, select **Web Page, Web Service, or EJB from J2C Java Bean**.
3. Click **Next**.
4. In the J2EE Resource from J2C Java Bean wizard, complete the following steps:
   a. On the J2C Java bean selection page, next to the J2C bean implementation entry field, click **Browse**.
   b. In the Select entries field, prime the entry field with an asterisk (*) to view the available data types, select **PBImpl.java** from the Matching types list and click **OK**.
   c. The J2C bean implementation field should contain */PhoneBookJ2CBean/sample/ims/PBImpl.java*.
5. Click **Next**.

   Note: If you get the message, "Resource *nnnnn*, referenced by the J2C code, is not found on servers. Please make sure that resource *nnnnn* exists," it may be that the resource has not been saved to the master configuration of the server. If you do not get this error, continue to Step 6.

   Otherwise, to eliminate this error message, select **Cancel**, then perform the following steps to save your resource to the master configuration of the server:

   a. In the Servers view, ensure that the server is started.
   b. Right-click the server and select **Run administrative console**.
   c. Log in to the administrative console.

d. In the left pane, expand **Resources** and select **Resource adapters**.
e. At this point a warning message may appear, "The master configuration has been updated. You currently have workspace conflicts with these modifications. To see these updates you must Save or discard your current workspace modifications." Save your modifications.
f. If the problem persists, try stopping and starting the server.

6. In the Deployment Information page, select **JSP** and then click **Next**.
7. In the JSP Creation page, select **Create a Faces JSP and add J2C Java bean as available page data**.
8. Next to Web project, click **New**, to create a new dynamic web project. The New Dynamic Web Project window opens.
9. In the Name field, type `PhoneBookWeb` and click **Finish**.
10. Select **Yes** to switch to the web perspective when requested to do so.
11. In the JSP Creation page, leave the JSP Folder name blank.
12. For the name of the Faces file, type `PBookF`.

    Note: The name of the EAR project defaults to *PhoneBookWebEAR*.
13. Click **Show Advanced** and type a name for the Resource Reference. For example, `myCFacRef`. **Note**: It is strongly recommended that you provide a resource reference for your J2EE resource. Not only does this allow you to map your J2EE resource to different J2C connection factories when you install your EAR on other WebSphere Application Servers, but if you do not use a resource reference you may receive unpredictable results when running your application.
14. Click **Finish**. The file, *PBookF.jsp* opens in the JSP editor.
15. Ensure that the **Design** tab in the JSP editor is selected.
16. Ensure that the **Page Data** view is open.
17. In the Page Data view, complete the following steps:
    a. Expand **java (sample.ims.PBImpl)**.
    b. Drag and drop the method *runPB(sample.ims.data.INPUTMSG)* onto the **Design** view of *PBookF.jsp* in the JSP editor. The *runPB(sample.ims.dat.INPUTMSG)* method entry in the Page Data view is identified with an **M** icon. If you have exposed properties of IMSInteractionSpec or IMSConnectionSpec, the signature of the runPB method will contain additional arguments.
    c. In the Configure Data Controls page of the Insert Java Bean wizard, select the fields that you wish to use to input data to the runPB method and optionally the exposed input properties of IMSInteractionSpec and IMSConnectionSpec. Then, click **Next**. You can change the order of the input fields by selecting a checked field and using the up and down arrows to the right of the Fields to display list to move the input field.
    d. Use the next page of the Insert Java Bean wizard to select the fields that you wish to see as output data from the output message of the runPB method, and optionally the exposed output properties of IMSInteractionSpec. Select the fields that you wish to see as output data from the runPB method and then click **Finish**.
    e. Close *PBookF.jsp* to save your changes.

You have now wrapped the J2C Java bean in a web page so that it can run on a J2EE application server.

Providing initial values for fields of a Faces JSP page

It is recommended that you provide initial values for the fields of a Faces JSP page that is generated by Rational Application Developer. To provide initial values, you must modify one of the methods of the generated Faces JSP. The method that returns the parameter bean which is used by the method that runs the IMS transaction must be updated.

It is recommended that you provide initial values for fields of a Faces JSP page because of the following reasons:

- The LL field of an IMS transaction input message must accurately reflect the size of the message buffer sent to IMS. Rather than leave the calcuation of this value to the user of the JSP page, you should initialize the field to the correct value using the getSize() method provided by the data binding for the transaction input message.
- The ZZ field of an IMS transaction input message does not affect the user of the JSP page and should be initialized to zero.
- The field for the transaction code of the input message should be initialized with the correct value. Generally, the user are not provided the transaction code for the IMS transaction that your application is running.
- Other fields may need to be initialized to remove blanks, making the Faces JSP easier to use.

In addition to initializing fields such as LL, ZZ, and the field for transaction code, you should hide these fields in the Faces JSP page because they do not affect the user. This topic does not discuss how to hide the fields of a JSP page.

To provide initial values for the field of a JSP page, you must modify a method of the generated Faces JSP. To modify the method of the generated Faces JSP, complete the following steps:

1. In the J2EE perspective of the Project Explorer view, expand **Dynamic Web Projects > PhoneBookWeb > Java Resources > JavaSource > pagecode > PBookF.java**.
2. In the PhoneBookWeb project, right-click **PBookF.java** and select **Open With > Java Editor**.
3. Update the method, **getJavaRunPBParamBean()**, with the following code:

```
public JavaRunPBParamBean getJavaRunPBParamBean() {
  if (javaRunPBParamBean == null) {
    javaRunPBParamBean = new JavaRunPBParamBean();
    // Initialize fields of input message.
     INPUTMSG input = javaRunPBParamBean.getArg();
    input.setIn__ll((short)input.getSize());
    input.setIn__zz((short)0);
    input.setIn__trcd("IVTNO");
    input.setIn__cmd("DISPLAY");
    input.setIn__name1("LAST1");
    input.setIn__name2("");
    input.setIn__extn("");
    input.setIn__zip("");
    // Initialize input fields for exposed input properties.
    javaRunPBParamBean.setMyImsRequestType(1);
    javaRunPBParamBean.setMyInteractionVerb(1);
    javaRunPBParamBean.setMyCommitMode(1);
    javaRunPBParamBean.setMyExecutionTimeout(0);
    javaRunPBParamBean.setMySocketTimeout(0);
 }
 return javaRunPBParamBean;
}
```

4. Save your changes and close the file.

The fields of INPUTMSG, which is the input message of the IMS transaction, are now initialized, as well as some exposed input properties.

Displaying javax.resource.ResourceException on a faces JSP page

Errors from the IMS resource adapter are returned to a Web application as exceptions of the type javax.resource.ResourceException. For example, if a Web application runs an IMS transaction and the transaction is stopped, a ResourceException subclass IMSDFSMessageException is thrown containing the IMS Connector for Java message ICO0079E. This message contains the `DFS065: TRAN/LTERM STOPPED` message from IMS.

To display the exceptions that are returned by the IMS resource adapter on the faces JSP generated by Rational Application Developer, you must modify the code of the Web application. To modify the code of the Web application, working with the *PhoneBookWeb* application, complete the following steps:

1. In the Project Explorer view of the J2EE perspective, expand **Dynamic Web Projects > PhoneBookWeb**.
2. Open `PBookF.java` in the Java editor.
3. In the PBookF.java file, locate the **doJavaRunPBAction()** method and modify the catch block as follows:

```
} catch (Exception e) {
            facesContext.addMessage(null,new FacesMessage(e.getLocalizedMessage()));
            logException(e);
        }
```

4. An error stating that the ″FacesMessage″ class cannot be resolved appears. You need to add an import statement in the PBookF.java file.
   a. In the Java editor, place your edit cursor on the **FacesMessages** class that is in error.
   b. Right-click and select **Source > Add Import**. The import statement is added to PBookF.java and the project automatically rebuilds. The error should disappear.
5. Save your changes and close the files.

You can now display the exceptions that are returned by the IMS resource adapter on the faces JSP generated by Rational Application Developer.

Using IMS data bindings in a CCI application

If you choose to write your Java application without using Rational Application Developer to generate a J2C Java bean and J2EE resource, you can still use the J2C option of Rational Application Developer to create Java data bindings for the input and output messages of your CCI application.

After you create the Java data binding for your IMS input and output messages, you can use those data bindings in a CCI application. The following steps explain how to use the data bindings in a simple CCI application:

1. From the menu bar, select **File > New > Project > Java project** and click **Next**.
2. Create a Java project named `SimpleCCIApp`.
3. Accept all other defaults and click **Finish**.
4. Click **Yes** to confirm the perspective switch when asked to switch to the Java Perspective.
5. Click **OK** to save the resource (`PBookF.jsp`).
6. In the Project Explorer view, expand **Other Projects** and right-click the Java project named *SimpleCCIApp* and then select **New > Package**.
7. In the New Java Package wizard, in the Name field enter `sample.ims` and click **Finish**.
8. In the Package view, right-click the project *SimpleCCIApp* and select **Properties > Java Build Path**.
   a. In the Projects tab, select the project containing the Java data bindings that you want your CCI application to use. For example, select the project, PhoneBookBindings that was created in the task, “Creating IMS Java data bindings” on page 9.
   b. In the Projects tab, select the project containing the IMS resource adapter that you want to use.

      If you do not have a project containing the IMS resource adapter that you want to use, you can import the RAR file for the IMS resource adapter into your workspace by completing the following steps:

      1) Exit the Java Build path wizard.
      2) In the Project Explorer view, right-click on your project and select **Import > File System**.
      3) Click **Next**.

4) In the File System wizard, click **Browse** next to the *From directory* field and choose your directory. You can get the RAR files for the IMS resource adapters from the following directories:
   - <RAD_install_dir>/Resource Adapters/ims for the JCA 1.0 IMS resource adapter
   - <RAD_install_dir>/Resouce Adapters/ims15 for the JCA 1.5 IMS resource adapter
5) Click on the box next to your directory to select it.
6) Click **Finish**.

c. In the Libraries tab, add the following JAR files to the build path for project *SimpleCCIApp* by clicking the **Add External JARs** button:
   - j2ee.jar
   - marshall.jar

   These JAR files are used by the Java data bindings generated by Rational Application Developer. The version of the jar files depends on the version of the IMS resource adapter that you selected. For example, if you selected the IMS resource adapter 9.1.0.1.1 and you installed the Test Environment for WebSphere Application Server Version 5, the jar files are located in the following directory path:
   - <RAD_install_dir>/runtimes/base_v5/lib

   If you selected the IMS resource adapter 9.1.0.2 and you installed the Test Environment for WebSphere Application Server Version 6, the jar files are located in the following directory path:
   - <RAD_install_dir>/runtimes/base_v6/lib

d. Click **OK**.

9. In the Package view, expand **Other Projects > SimpleCCIApp**, right-click the package **sample.ims**, and select **New > Class**.
10. In the Java Class wizard, complete the following steps:
    a. In the name field, enter CCIApp for the name of the new class.
    b. In the *Which method stubs would you like to create?* option, ensure that the **public static void main(String{}args)** and **Inherited abstract methods** check boxes are selected and click **Finish**.
11. Edit the CCIApp.java source. Copy the following sample code and paste into the file:

```
/*
 *
 * TODO To change the template for this generated file go to
 * Window - Preferences - Java - Code Style - Code Templates
 */
package sample.ims;

import com.ibm.connector2.ims.ico.*;
import javax.resource.cci.*;

/**
 *
 *
 * TODO To change the template for this generated type comment go to
 * Window - Preferences - Java - Code Style - Code Templates
 */
public class CCIApp {

    public static void main(String[] args) {

        Connection conn = null;

        try{
            IMSManagedConnectionFactory mcf = new IMSManagedConnectionFactory();
            mcf.setHostName("yourHostName");
            mcf.setPortNumber(new Integer(0));
            mcf.setDataStoreName("yourDataStoreName");
```

```
            ConnectionFactory cf = (ConnectionFactory) mcf.createConnectionFactory();
            IMSConnectionSpec cSpec = new IMSConnectionSpec();

            conn = cf.getConnection(cSpec);

            Interaction interAction = conn.createInteraction();
            IMSInteractionSpec iSpec = new IMSInteractionSpec();
            iSpec.setInteractionVerb(1);        // SEND_RECEIVE
            iSpec.setImsRequestType(1);            // TRANSACTION
            iSpec.setCommitMode(1);                // SEND_THEN_COMMIT

            sample.ims.INPUTMSG input = new INPUTMSG();
            input.setIn__ll((short) input.getSize());
            input.setIn__zz((short) 0);
            input.setIn__trcd("IVTNO");
            input.setIn__cmd("DISPLAY");
            input.setIn__name1("LAST1");

            sample.ims.OUTPUTMSG output = new sample.ims.OUTPUTMSG();
            interAction.execute(iSpec, input, output);

            System.out.println(
                    "Output message is... " +
                    "\nMSG: " + output.getOut__msg() +
                    "\nNAME1: " + output.getOut__name1() +
                    "\nNAME2: " + output.getOut__name2() +
                    "\nEXTN: " + output.getOut__extn() +
                    "\nZIP: " + output.getOut__zip()
            );
        }
        catch(Exception e)
        {
            System.out.println("Caught exception is: " + e.getMessage());
        }
    }
}
```

The CCIApp.java is a simple two-tier, non-managed Java application program. It uses the Java data bindings, sample.ims.INPUTMSG and sample.ims.OUTPUTMSG, that were created by the CICS/IMS Java Data Binding wizard in Rational Application Developer. Edit CCIApp.java and modify the values used by the setHostName(), setPortNumber(), and setDataStoreName() statements for your environment.

12. Click **File > Save**.
13. To run your Java application, in the Project Explorer view, expand **Other Projects > SimpleCCIApp**.
14. Right-click **CCIApp.java** and select **Run > Java Application**. The following information is displayed in the Console view:

```
Output message is...
MSG: ENTRY WAS DISPLAYED
NAME1: LAST1
NAME2: FIRST1
EXTN: 8-111-1111
ZIP: D01/R01
```

Chapter 4. Running your web application

The topics in this section describe how to run your web application. There are a number of environments in which you can run your application. You can use the Test Environment in Rational Application Developer to test your application. Another option is you can export your application to a standalone instance of WebSphere Application Server. The topics included are:

Running your web application using the Rational Application Developer test environment

You can test your application using the test environment in Rational Application Developer.

Rational Application Developer includes a number of optional test environments. For example, you can include test environment for WebSphere Application Server Version 6.0 and test environments for legacy application servers such as, WebSphere Application Server Version 5.1 and WebSphere Application Server Version 5.0.

The test environment that you use to run your web application depends on how the application is generated. For example, if you selected servlet version 2.4 (J2EE Version 1.4) when you defined the dynamic web project for your application, you must select WebSphere Application Server Version 6.0 as your target server and you must test your application using the WebSphere Application Server Version 6.0 Test Environment.

To run your application using the Rational Application Developer test environment, complete the following steps:

1. In the Project Explorer view, expand **Dynamic Web Projects > PhoneBookWeb > WebContent**.
2. Right-click **PBookF.jsp** in the Web Content folder and select **Run > Run on server**.
3. In the Server Selection wizard, select **Choose an existing server**.
4. Select the server instance you configured. For example, **WebSphere 6.0 Server@localhost** and then click **Next**.
5. In the Add and Remove Projects page, ensure that your new EAR project, *PhoneBookWebEAR*, is in the list of **Configured projects** and then click **Finish**.
6. The faces JSP, *PBookF.jsp*, displays in the web browser. Enter values in the input fields; for example:
   - In__trcd: IVTNO
   - In__zz: 0
   - In__name1: LAST1
   - In__cmd: DISPLAY
   - In__ll: 59
7. For all the remaining input fields, ensure that trailing blanks are removed and then click **Submit**.
8. The following output is displayed:

```
Out__ll:  93
Out__zz:  768
Out__msg:  ENTRY WAS DISPLAYED
Out__name1: LAST1
Out__name2: FIRST1
Out__extn:  8-111-1111
Out__zip:  D01/R01
```

Note: If you exposed IMSInteractionSpec output properties such as:

- convEnded

- asyncOutputAvailable
- mapName

as data, they will also display in the output.

You have completed testing your application in the test environment of the Rational Application Developer.

Running your application in a standalone WebSphere Application Server

You can test your application using a standalone WebSphere Application Server. To run your application in a standalone WebSphere Application Server, you must first export your web application from Rational Application Developer as an Enterprise Application Archive (EAR) file and then install the EAR file on the server.

If you do not package the IMS resource adapter with your enterprise application, you must ensure that the IMS resource adapter that your application uses is installed on the standalone WebSphere Application Server. You also need to ensure that a connection factory has been defined to create connections to the IMS Connect and IMS that you wish to run the IMS transaction.

The following topics describe how to use a standalone WebSphere Application Server to run your application:

Exporting your application as an EAR file

Before you install and run your application on a standalone WebSphere Application Server, you must first export your application from Rational Application Developer as an EAR file.

To export your application as an EAR file, complete the following steps in Rational Application Developer:

1. From the menu bar, select **File > Export**.
2. From the **Select and export destination** list, select **EAR file**.
3. Click **Next**.
4. From the **EAR project** drop down list, select **PhoneBookWebEAR**.
5. From the **Destination** field, click **Browse** to choose the location where you want to save the EAR file.
6. Click **Finish**.

Installing the IMS resource adapter in WebSphere Application Server

Before you use the EAR file to install your application, you must install the IMS resource adapter in WebSphere Application Server, if it is not already installed.

Note: You can only have one standalone version of the IMS RAR file installed on WebSphere Application Server because the different versions of the IMS RAR files share the same classloader. If there are two versions of the IMS RAR installed, there will be a conflict because they share the same class names.

To install the IMS RAR file, complete the following steps:

1. Ensure that WebSphere Application Server is started.
2. On the Welcome page, in the left-hand navigation, expand **Resources** and select **Resource Adapters.** The Resource Adapter page opens.
3. On the resource adapter page, scroll to the bottom and click the **Install RAR** button. This takes you to the Install RAR File page.

4. Click **Browse** to navigate to the path where the RAR file is installed and select **Next**.
5. In the next page, type in a name for your RAR file, for example, IMSRAR9102, and click **OK**.
6. Save your changes by clicking **Save** at the top of the page.
7. Click **Save** again when prompted to save your workspace to the master configuration.

The IMS RAR file is now installed on your WebSphere Application Server.

Creating a connection factory for the IMS resource adapter

After you have installed the IMS resource adapter on your server, you need to create a connection factory for the IMS resource adapter, if it does not already exist. The connection factory is used by one or more application to create connections between IMS and IMS Connect.

To create a connection factory to for the IMS resource adapter, complete the following steps:

1. On the welcome page of WebSphere Application Server, expand **Resources > Resource Adapters**.
2. Scroll to the bottom of the page and click the name of the RAR file for which you want to create a connection factory. For example, select **IMSRAR9102**. The General Properties page will appear.
3. In the General Properties page, in the right-hand column under Additional Properties, select **J2C Connection Factories**.
4. Click the **New** button and in the Name field type a name for the connection factory. For example, myIMSA.
5. In the JNDI name field, accept the default.
6. Click **OK**.
7. Click the name of the connection factory that you just created, *myIMSA*. The configuration page opens.
8. On the configuration page, in the right-hand column under Additional Properties, click **Custom properties**.
9. Click on each of the fields below to fill in the values to configure your connection factory. For example:
   - In the **HostName** row, click on the existing value. In the configuration page that comes up, type MYHOST.MYCOMPANYNAME.COM in the **Value** field and then click **OK**.
   - In the **PortNumber** row, click on the existing value. In the configuration page that comes up, type 9999 in the **Value** field and then click **OK**.
   - In the **DataStoreName** row, click on the existing vaule. In the configuration page that comes up, type MYDSTOR in the **Value** field and then click **OK**.
10. Click **Save** at the top.
11. Click **Save** again when prompted to save your workspace to the master configuration.
12. Stop the server and then restart it to see the new connection factory.

You now have created a connection factory for your RAR file.

Installing your EAR file in WebSphere Application Server

After you exported your application as an EAR file, installed an IMS resource adapter and created a connection factory, you can use the EAR file that you created to install the application in WebSphere Application Server and then run the application.

To install your EAR file and run your application, complete the following steps:

1. Ensure that the server is started.
2. On the welcome page of WebSphere Application Server, expand **Applications** and select **Install New Application**.
3. Click **Browse** to navigate to the EAR file that you want to install and then click **Next**.

4. Click **Next** again until the page **Install New Application**, which lists the steps for installation, appears
5. Click **Step 4: Map resource references to resources**.
6. In the javax.resource.cci.Connection.Factory portion of the page, follow the steps provided on the page. Ensure that you complete the steps described on the page. For example:
   a. Select an existing resource JNDI name in the drop down list.
   b. Scroll down the page and select the resource references that you want to map.
   c. Then scroll back up the page and click **Apply**.
7. Click **Step 7: Summary** and click **Finish**.
8. Click **Save** when prompted to save your workspace to the master configuration. After you save installing your EAR file, the welcome page opens.
9. On the welcome page, expand **Applications > Enterprise Application**.
10. Select the checkbox of the new application that you just installed and click **Start** to run your application.

You have now installed your application on a production WebSphere Application Server.

Chapter 5. Configuring your application

The topics in this section describe how to configure your application for your service. The topics included are:

Execution timeout

The *execution timeout value* for the IMS resource adapter is defined as the maximum amount of time allowed for IMS Connect to send a message to IMS and receive a response from IMS. For details about the execution timeout value, see Setting execution timeout values and Valid execution timeout values.

Before the introduction of the executionTimeout property, you were limited to setting a timeout value on a global level, which was specified in the IMS Connect configuration file. Every interaction between IMS Connect had the same timeout value.

With the executionTimeout property, you can set individual timeout values on a per interaction basis rather than on a global basis. If an interaction isn't complete before timeout occurs, IMS Connect returns an error message to the IMS resource adapter. The IMS resource adapter returns an exception indicating that the duration of time for IMS to respond to IMS Connect has exceeded the execution timeout value.

Note: Because the connection between the IMS resource adapter and IMS Connect is persistent, when execution timeout occurs, the socket is not closed. Instead, the socket is available for reuse for subsequent interactions..

Execution timeout in conversational transactions

In a conversational transaction, the execution timeout value applies to each iteration of that conversation. An *iteration* consists of one input message sent to IMS and one output message received from IMS. If one iteration of the conversation times out, the entire conversation ends.

Execution timeout exceptions

If a valid execution timeout value is specified for a particular interaction and execution timeout occurs, the Java application submitting the interaction receives the exception javax.resource.spi.EISSystemException. If you specify an invalid execution timeout value, the exception javax.resource.NotSupportedException is thrown when execution timeout occurs.

Valid execution timeout values

The execution timeout value is represented in milliseconds and must be a decimal integer in the range of 1 to 3600000, inclusively. That is, the executionTimeout value must be greater than zero and less than or equal to one hour. The execution timeout value can also be -1 if you want an interaction to run without a time limit. The execution timeout value cannot contain non-numeric characters.

If you do not specify an execution timeout value or if the value that you specify is invalid:

- For SYNC_SEND_RECEIVE interactions, the timeout value in the IMS Connect configuration member is used and the interaction continues to run.
- For SYNC_RECEIVE_ASYNCOUTPUT, SYNC_RECEIVE_ASYNCOUTPUT_SINGLE_NOWAIT, and SYNC_RECEIVE_ASYNCOUTPUT_SINGLE_WAIT interactions, IMS Connect will set the timeout value to two seconds and the interaction continues to run.

Additionally, if you specify an invalid value, the exception javax.resource.NotSupportedException is thrown when timeout occurs for that interaction.

Tip: The host system administrator determines the global timeout value in the IMS Connect configuration member. To display this value, issue the VIEWHWS command on the MVS console. See the *IMS Connect User's Guide and Reference* (SC27-0946-03) for more information on the VIEWHWS command.

If a valid execution timeout value is set, this value is converted into a value that IMS Connect can use. The following table describes how the values you specify are converted to the values that IMS Connect uses:

Range of user-specified values	Conversion rule
1 - 250	If the user-specified value is not divisible by 10, it is converted to the next greater increment of 10.
251 - 1000	If the user-specified value is not divisible by 50, it is converted to the next greater increment of 50.
1001 - 60000	The user-specified value is converted to the nearest increment of 1000. Values that are exactly between increments of 1000 are converted to the next greater increment of 1000.
60001 - 3600000	The user-specified value is converted to the nearest increment of 60000. Values that are exactly between increments of 60000 are converted to the next greater increment of 60000.

For example, if you specify a value of 1, this value is converted to 10 (because 1 is not divisible by 10 and 10 is the next increment that is greater than 1). The following examples illustrate how the conversion works for each range of values:

User-specified value (milliseconds)	Converted value (milliseconds)
1	10
11	20
251	300
401	450
1499	1000
1500	2000
60000	60000
89999	60000
3600000	3600000
3750000	3600000

Setting execution timeout values

executionTimeout is a property of the IMSInteractionSpec class. The execution timeout value that you set is converted to a value that IMS Connect uses. This conversion occurs to meet the requirements of IMS Connect. **Important:** Other timeout values can affect your interactions. If other timeout values are less than the execution timeout value you set for your IMS interaction, these other timeout values can cause the interaction to expire. Other timeout values include:

- Connection timeout property of J2C connection factories

- EJB transaction timeout value
- Browser timeout value
- Servlet HTTP session or EJB session timeout values

For example, when WebSphere Application Server is running on the z/OS platform, the server consists of two parts, a controller and a set of one or more servants. Application work is dispatched into servant regions. Application work is, by default, timed. In general, when an application in dispatch reaches its timeout, the servant region where it is dispatched is abended and restarted. The server stays up and continues taking work. For this reason, you should use care when choosing execution timeout values that are greater than WebSphere Application Server timeout values, or when choosing the execution timeout value of -1 (wait forever). In addition, if you are planning on disabling WebSphere Application Server timeouts, you should check the server documentation in order to better understand the implications of doing this.

Another example is if you configure the execution timeout value to be greater than the timeout value specified for a browser response, then the execution timeout value is never used because the browser timeout value is exceeded first.

You can provide a value for the executionTimeout property of an IMSInteractionSpec class in one of two ways:

- Using Rational Application Developer
- Using the setExecutionTimeout method

With the first method, using Rational Application Developer, you can set the execution timeout value when you initially define the IMS binding properties for a new J2C Java Bean.

To edit the IMS binding properties that are already defined for a new J2C Java Bean, complete the following steps:

1. Open the appropriate IMS binding Java file using the Java Editor.
2. Locate the doclet tag for the IMSInteractionSpec class.
3. Modify the doclet tag to add executionTimeout property, if it is not listed and specify a value for it. If it is listed, modify the value.
4. Close the editor and click **Yes** to save your changes.

Note: You can also code individual timeout values for different interactions using the method described below in Exposing the executionTimeout property of the IMSInteractionSpec and Using the setExecutionTimeout method. If you code an execution timeout value in your Java client application code, that value overrides any execution timeout value that you set in Rational Application Developer.

With the second method, you can use the setExecutionTimeout method to set an execution timeout value. If you are creating a CCI application, you will have access to the setExecutionTimeout method of the IMSInteractionSpec. To use the setExecutionTimeout method, you need to instantiate a new IMSInteractionSpec or obtain the IMSInteractionSpec from your specific interaction. Then, set the executionTimeout value for the IMSInteractionSpec by using the setExecutionTimeout method provided by the IMSInteractionSpec class. For example:

```
interactionSpec.setExecutionTimeout(timeoutValue);
```

After you set the executionTimeout value for the IMSInteractionSpec, assign this interactionSpec to the specific interaction.

Socket timeout

Socket timeout is the maximum amount of time IMS Connector for Java will wait for a response from IMS Connect before disconnecting the socket and returning an exception to the client application.

If there are network problems or routing failures, the socketTimeout property prevents a hang in the system where the client using the IMS resource adapter is waiting indefinitely for a response from IMS Connect. Because the socketTimeout property is based on the TCP/IP sockets with which IMS Connect and the IMS resource adapter use to communicate, the socketTimeout property is not applicable with Local Option.

With the socketTimeout property, you can set individual timeout values for a particular interaction using a socket. The value, in milliseconds, can be set on the socketTimeout property in IMSInteractionSpec. If the socketTimeout property is not specified for an interaction or it is set to zero milliseconds, this means there is no socket timeout and the connection will wait indefinitely. The default socket timeout value is zero.

When determining the Socket Timeout value, other existing timeout values should be taken into account. For example, browser session timeout value, Execution Timeout, EJB transaction timeout value, WebSphere Application Server connection timeout value, and HTTP session timeout value used by servlets and stateful session beans.

If a valid socket timeout value is specified for a particular interaction and socket timeout occurs, a *java.io.IOInterruptedException* is thrown and the J2EE JCA exception, *javax.resource.spi.CommException* is raised. The J2EE JCA exception message indicates that the client has spent more time than was allocated by the socketTimeout value to communicate with IMS Connect.

Setting the Socket Timeout Value

When setting the socketTimeout value, you need to consider the executionTimeout value. The executionTimeout property is the maximum amount of time allowed for IMS Connect to send a message to IMS and receive a response from IMS. The socketTimeout value encapsulates the executionTimeout value. Therefore, the socketTimeout value should be greater than the executionTimeout property because the socket may time out unnecessarily if its value is set to less than the executionTimeout value. The following table lists suggested values for socketTimeout based on executionTimeout values.

Execution Timeout Value (milliseconds)	**Execution Timeout Behavior**	**Suggested Socket Timeout Value**
0 (or no value)	The default value from the IMS Connect configuration file is used.	The socket timeout value should be greater than the execution timeout default value specified in the IMS Connect configuration file.
1 - 3,6000,000	The wait response times out after the specified millisecond value.	The socket timeout value should be greater than the execution timeout value.
-1	The wait response is indefinite.	Set the socket timeout value to 0 so that the connection waits indefinitely.

There are two ways to set the socket timeout value. You can either write an application using the JCA Common Client Interface (CCI) to access the getter and setter methods provided with the IMSInteractionSpec or use the tooling provided by WebSphere Studio Application Developer Integration Edition.

Using the CCI application to set a socket timeout value

If you are creating a CCI application, you will have access to the setSocketTimeout method of the IMSInteractionSpec. To use the setSocketTimeout method, you need to instantiate a new IMSInteractionSpec or obtain the IMSInteractionSpec from your specific interaction. Then set the socketTimeout value for the IMSInteractionSpec by using the setSocketTimeout method provided by the IMSInteractionSpec class. For example:

```
interactionSpec.setSocketTimeout(timeoutValue1);
interaction.execute(interactionSpec,input,output);

interactionSpec.setSocketTimeout(timeoutValue2);
interaction.execute(interactionSpec,input,output);
```

Using WebSphere Studio Application Developer Integrated Edition to set a socket timeout value

You can use WebSphere Studio Application Developer Integrated Edition to set the socket timeout value when you initially define the operation binding properties for an IMS service. To edit the operation binding properties that are already defined for an IMS service, complete the following steps:

1. Open the appropriate IMS binding WSDL file using the WSDL Editor.
2. In the Bindings container of the Graph view, expand the IMS binding WSDL file and expand the appropriate binding operation file.
3. Select the **operation extensibility element** (for example, **ims:operation**) and edit the values of the properties in the property table.
4. Select the operation extensibility element again to indicate that changes have been made.
5. Close the editor and click **Yes** to save your changes.

Connection properties

When you create an IMS service definition or define an IMS connection factory to WebSphere Application Server, you must provide values for certain properties of the connection between IMS Connector for Java and IMS Connect. The following list describes these connection properties:

Host name
Mandatory for TCP/ IP connections: The IP address or host name of the machine running the target IMS Connect. You must replace the value ″myHostNm ″ with a value that is valid for your IMS environment.

Port number
Mandatory for TCP/IP connections: The number of a port used by the target IMS Connect for TCP/IP connections. Multiple sockets can be open on a single TCP/ IP port. See ″Configuring IMS Connect″ in the *IMS Connect Guide and Reference* (SC27-0946-03) for additional information about the PortNumber property. You must replace the value of ″0″ with a value that is valid for your IMS environment.

CM0Dedicated
The default is false. A value of FALSE indicates the connection factory will generate shareable persistent socket connections and IMS Connector for Java will generate a clientID to identify the socket connection. These connections can be used by commit mode 0 and commit mode 1 interactions. A value of TRUE indicates the connection factory will generate dedicated persistent socket connections, which require user-specified clientIDs to identify the socket connections. A dedicated persistent socket connection is reserved for a particular clientID and only commit mode 0 interactions are allowed. This property applies to TCP/IP connections only.

SSL Enabled
The default is false. This property is only valid for TCP/IP connections. A value of true indicates that IMS Connector for Java will create an SSL socket connection to IMS Connect using the HostName and PortNumber specified in these connection properties. This port must be configured as an SSL port by IMS Connect. A value of false indicates that SSL sockets will not be used for connecting to the port specified in the Port Number property.

KeyStore Name

For non-z/OS platforms, specify the fully-qualified path name of your JKS keystore file. For z/OS, specify the name of your JKS keystore file as above, or a special string that provides the information needed to access your RACF keyring.

Private keys and their associated public key certificates are stored in password-protected databases called keystores. For convenience, trusted certificates can also be stored in the keystore and then the Truststore Name property can either be empty or could point to the keystore file. If the TrustStore Name/TrustStore Password property is left empty, an informational message is generated in the server log.

The keystore name can be used to specify either a JKS keystore or a RACF keyring when running on z/OS. An example of a fully-qualified path name of your JKS keystore file is *c:\keystore\MyKeystore.ks*. A RACF keyring is specified as: *keystore_type:keyring_name:racfid*. The keystore_type must be either **JCERACFKS** when software encryption is used for SSL or **JCE4758RACFKS** if hardware encryption is used. Replace *keyring_name* with the name of the RACF keyring that you are using as your keystore and *racfid* with a RACF ID that is authorized to access the specified keyring. Examples of RACF keyring specifications are `"JCERACFKS:myKeyring:kruser01"` or `JCE4758RACFKS:myKeyring:kruser01"`. When running in z/OS, if the keystore name matches the above RACF keyring format, IMS Connector for Java will use the specified RACF keyring as its keystore. If the keystore type specified is anything other than **JCERACFKS** or **JCE4758RACFKS**, IMS Connector for Java attempts to interpret the keystore name specified as the name of a JKS keystore file.

Note: The JKS file can have other file extensions; it does not have to have to be .ks.

KeyStore Password

Specify the password for the keystore. Private keys and their associated public key certificates are stored in password-protected databases called keystores.

TrustStore Name

For non-z/OS platforms, specify the fully-qualified path name of your JKS truststore file. For z/OS, specify the JKS name or the RACF keyring of the truststore. The same format is used for the values of the Keystore Name and Truststore Name properties. See the description of the Keystore Name property for a discussion of this format.

A truststore file is a key database file (keystore) intended to contain public keys or certificates. For convenience, private keys can also be stored in the Truststore and then the Keystore Name property can either be empty or could point to the truststore file. If the KeyStore Name/KeyStore Password property is left empty, an informational message will be generated in the server log.

Note: The JKS file can have other file extensions; it does not have to have to be .ks.

TrustStore Password

Specify the password for the truststore. A truststore file is a key database file that contains public keys.

Encryption Type

Select the encryption type. Strong and weak are related to the strength of the ciphers, that is, the key length. All those ciphers that can be used for export come under the weak category and the rest go into the strong category. By default, the encryption type is set to weak.

IMS Connect name

Mandatory for Local Option connections: The job name of the target IMS Connect. If the IMS Connect name is specified, it overrides the Host name, Port number, and SSL-related properties.

Default user name

Optional: The default security authorization facility (SAF) user name that will be used for connections created by this connection factory if no UserName property is provided by the application component.

Default password
Optional: The password that will be used for connections created by this connection factory if the default user name is used.

Default group name
Optional: The IMS group name that will be used for all connections created by this connection factory if the default user name is used.

Note: The GroupName property can only be provided in a component-managed environment.

Data store name
Mandatory: The name of the target IMS datastore. It must match the ID parameter of the Datastore statement that is specified in the IMS Connect configuration member. It also serves as the XCF member name for IMS during internal XCF communications between IMS Connect and IMS OTMA. You must replace the default value ″myDStrNm″ with a value that is valid for your IMS environment.

Trace level
Optional: The level of information to be traced. For additional information on trace level, see Logging and tracing with the IMS resource adapter.

TransactionResourceRegistration
Optional: The type of transaction resource registration (enlistment). Valid values are either ″static″ (immediate) or ″dynamic″ (deferred). If this property is set to ″dynamic″, the enlistment of the resource to the transaction scope will be deferred until the resource is used for an interaction for the first time.

MFS XMI Repository ID
A resource property of a defined J2C Connection Factory, which is accessible on the J2C options page of the server configuration. This field contains a unique name for identifying the repository location. This ID must match the repository field defined in the generated format handler of your application. The default for this field is ″default″.

MFS XMI Repository URI
A resource property of a defined J2C Connection Factory, which is accessible on the J2C options page of the server configuration. This field specifies the physical location of the XMI repository. Valid formats for this field include:

- `file://path_to_xmi`, where path_to_xmi is a directory on the local file system containing the xmi files, for example `file://c:/xmi`.
- `http://url_to_xmi`, where url_to_xmi is a valid url that resolves to a directory containing the xmi files, for example `http://sampleserver.com/xmi`.
- `hfs://path_to_xmi` where path_to_xmi is the HFS directory on the host z/OS. This format is only supported for WebSphere Application Server for z/OS.

IMSInteractionSpec properties

When you define a Java method for a J2C Java Bean, you must provide values for the properties that describe the interaction with IMS that will be performed by the Java method. These are values of the input properties of IMSInteractionSpec. The following list describes all output properties as well as input properties of IMSInteractionSpec, including those that are not set by the application component:

asyncOutputAvailable
This is an output only property. It can be used by a Java application to determine if there is queued output for the TPIPE associated with the connection used for a commitMode 0 interaction. For dedicated persistent socket connections, this is the value in the clientID property of IMSConnectionSpec. For shareable persistent socket connections, this is value generated by IMS Connector for Java. The value of asyncOutputAvailable is true if there are messages in the queue. The asyncOutputAvailable property is not set on input by the application component.

Note: If your Java application uses this property, it must be exposed as an output property of IMSInteractionSpec. See Creating an application to run a Commit mode 0 transaction for information on exposing the properties of IMSInteractionSpec.

convEnded

This is an output only property. It can be used by a Java application to determine if a conversation has ended (true). The convEnded property is not set on input by the application component. **Note:** If your Java application uses this property, it must be exposed as an output property of IMSInteractionSpec. See Creating an application to run a commit mode 0 transaction for information on exposing the properties of IMSInteractionSpec.

commitMode

Used by the IMS resource adapter to indicate the type of commit mode processing to be performed for an IMS transaction. See Overview of commit mode processing for more information. The commitMode property can be set to 0 or 1 when interactionVerb is set to SYNC_SEND_RECEIVE. When interactionVerb is set to SYNC_RECEIVE_ASYNCOUTPUT, SYNC_RECEIVE_ASYNCOUTPUT_SINGLE_NOWAIT, SYNC_RECEIVE_ASYNCOUTPUT_SINGLE_WAIT, or SYNC_SEND, IMS Connector for Java uses commitMode 0. commitMode 1 is required when interactionVerb is set to SYNC_END_CONVERSATION.

If commitMode is 0 and a shareable persistent socket is used for the interaction, the clientID must not be specified. If commitMode 0 is specified for an interaction on a shareable persistent socket, the output message from a transaction can be purged or rerouted. The undelivered secondary output from a program to program switch can also be purged or rerouted.

If a dedicated persistent socket connection is used for an interaction, the commitMode must be 0 and the clientID property of the IMSConnectionSpec used for the connection must be provided. If a dedicated persistent socket is used for a commitMode 0 interaction, undelivered output messages are always recoverable and cannot be purged or rerouted.

socketTimeout

The maximum amount of time IMS Connector for Java will wait for a response from IMS Connect before disconnecting the socket and returning an exception to the client application. The socketTimeout value is represented in milliseconds. To use socket timeout, the value must be greater than zero. If a socket timeout is not specified for an interaction or it is supplied with a socket timeout value of zero milliseconds, this will result in no socket timeout or an infinite wait. For more information see Socket timeout and Setting socket timeout values.

executionTimeout

The maximum amount of time allowed for IMS Connect to send a message to IMS and receive a response. The executionTimeout value is represented in milliseconds and must be a decimal integer that is either -1 or between 1 and 3,600,000, inclusively. That is, the executionTimeout value must be greater than zero and less than or equal to one hour. If a -1 value is set for this property, the interaction will run without a time limit. For more information, see Execution timeout, Setting execution timeout values, and Valid execution timeout values.

imsRequestType

Indicates the type of IMS request and determines how output from the request is handled by the IMS resource adapter. Integer values are:

Value	Named constant in IMSInteractionSpecProperties	Description
1	IMS_REQUEST_TYPE_IMS_ TRANSACTION	The request is an IMS transaction. Normal transaction output returned by IMS is used to populate the application's output message. If IMS returns a ″DFS″ message, the IMS resource adapter throws an IMSDFSMessageException containing the "DFS" message. This value for imsRequestType is used for applications that are not generated using WebSphere Studio MFS support.
2	IMS_REQUEST_TYPE_IMS_COMMAND	The request is an IMS command. Command output returned by IMS, including ″DFS″ messages, is used to populate the application's output message. The IMSDFSMessageException is not thrown. This value for imsRequestType is used for applications that submit IMS commands.
3	IMS_REQUEST_TYPE_MFS_ TRANSACTION	This value for imsRequestType is reserved for applications that are generated using WebSphere Studio MFS support. Normal transaction output returned by IMS, as well as ″DFS″ messages, are used to populate the application's output message. The IMSDFSMessageException is not thrown.

interactionVerb

The mode of interaction between the Java application and IMS. The values currently supported by the IMS resource adapter are:

Value	Named constant in IMSInteractionSpecProperties	Description
0	SYNC_SEND	The IMS resource adapter sends the client request to IMS through IMS Connect and does not expect a response from IMS. With a SYNC_SEND interaction, the client does not need to synchronously receive a response from IMS. SYNC_SEND is supported on both shareable and dedicated persistent socket connections and is only allowed with commitMode 0 interactions. If the interactionVerb is set to SYNC_SEND, execution timeout and socket timeout values are ignored. Note: imsRequest type 2 is not allowed with SYNC_SEND and will generate an exception.

Value	Named constant in IMSInteractionSpecProperties	Description
1	SYNC_SEND_RECEIVE	The execution of an IMS Interaction sends a request to IMS and receives a response synchronously. A typical SYNC_SEND_RECEIVE interaction is the running of a non-conversational IMS transaction in which an input record (the IMS transaction input message) is sent to IMS and an output record (the IMS transaction output message) is returned by IMS. SYNC_SEND_RECEIVE interactions are also used for the iterations of a conversational IMS transaction. A conversational transaction requires commitMode 1. A non-conversational transaction can run using either commitMode 1 or commitMode 0. If commitMode 0 is used on a dedicated persistent socket, a value for the clientID property of IMSConnectionSpec must be provided. If commitMode 0 is used on a shareable persistent socket, a value for the clientID property of IMSConnectionSpec must **not** be provided.
3	SYNC_END_CONVERSATION	If the application executes an interaction with interactionVerb set to SYNC_END_CONVERSATION, the IMS resource adapter sends a message to force the end of an IMS conversational transaction. The IMSInteractionSpec property, commitMode, and the IMSConnectionSpec property, clientID, do not apply when SYNC_END_CONVERSATION is provided for interactionVerb.
4	SYNC_RECEIVE_ASYNCOUTPUT	interactionVerb SYNC_RECEIVE_ASYNCOUTPUT is valid on both shareable persistent and dedicated persistent socket connections. SYNC_RECEIVE_ASYNCOUTPUT is used to retrieve asynchronous output that was not delivered. When SYNC_RECEIVE_ASYNCOUTPUT is used on a dedicated persistent socket, a value must be provided for the clientID property of IMSConnectionSpec. A SYNC_RECEIVE_ASYNCOUTPUT interaction on a shareable persistent socket connection must be in the same application as the original SYNC_SEND or SYNC_SEND_RECEIVE interaction and must use the same shareable persistent connection. This primarily occurs following execution timeout. With this type of interaction, the Java client can only receive a single message. If there are no messages in the IMS OTMA Asynchronous Queue for the clientID when the request is made, no further attempts are made to retreive the message. No message is returned and a timeout will occur after the length of time specified in the executionTimeout property of the SYNC_RECEIVE_ASYNCOUTPUT interaction.

Value	Named constant in IMSInteractionSpecProperties	Description
5	SYNC_RECEIVE_ASYNCOUTPUT_ SINGLE_NOWAIT	interactionVerb SYNC_RECEIVE_ASYNCOUTPUT_ SINGLE_NOWAIT is valid on both shareable and dedicated persistent socket connections. It is used to retrieve asynchronous output. A SYNC_RECEIVE_ASYNCOUTPUT_ SINGLE_NOWAIT interaction on a shareable persistent socket connection must be in the same application as the original SYNC_SEND or SYNC_SEND_RECEIVE interaction and must use the same shareable persistent connection. This primarily occurs following execution timeout. With this type of interaction, the Java client can only receive one single message. If there are no messages in the IMS OTMA Asynchronous Queue for the clientID when the request is made, no further attempts will be made to retrieve the message. No message will be returned and a timeout will occur after the length of time specified in the executionTimeout property of the SYNC_RECEIVE_ASYNCOUTPUT_ SINGLE_NOWAIT interaction. **Note**: The interactionVerbs, SYNC_RECEIVE_ASYNCOUTPUT and SYNC_RECEIVE_ASYNCOUTPUT_ SINGLE_NOWAIT , perform the same function. However, it is recommended to use SYNC_RECEIVE_ASYNCOUTPUT_ SINGLE_NOWAIT

Value	Named constant in IMSInteractionSpecProperties	Description
6	SYNC_RECEIVE_ASYNCOUTPUT_ SINGLE_WAIT	interactionVerb SYNC_RECEIVE_ASYNCOUTPUT_ SINGLE_WAIT is used to retrieve asynchronous output. It is valid on both shareable and dedicated persistent socket connections. A SYNC_RECEIVE_ASYNCOUTPUT_ SINGLE_WAIT interaction on a shareable persistent socket connection must be in the same application as the original SYNC_SEND or SYNC_SEND_RECEIVE interaction and must use the same shareable persistent connection. This primarily occurs following execution timeout. With this type of interaction, the Java client can only receive one single message. If there are no messages in the IMS OTMA Asynchronous Queue for the clientID when the request is made, IMS Connect waits for OTMA to return a message. IMS Connect waits the length of time specified in the executionTimeout property of the SYNC_RECEIVE_ASYNCOUTPUT_ SINGLE_WAIT interaction before returning an exception.

The J2EE Connection Architecture (JCA) values SYNC_RECEIVE (2) is not currently supported.

ltermName

The LTERM name used to override the value in the LTERM field of the IMS application program's I/O PCB. See the *IMS Connect User's Guide and Reference* (SC27-0946-23) for a description of how to use the LTERM override.

The value of this property can be set if the client application wants to provide an LTERM override name. This name will be in the IMS application program's I/O PCB, with the intent that the IMS application will make logic decisions based on this override value.

mapName

The mapName field typically contains the name of a Message Format Service (MFS) control block. MFS is the component of IMS that performs online formatting of transaction input and output messages. Since IMS Connect uses IMS OTMA to access IMS, MFS online formatting is bypassed. However, the mapName field can still be used by a Java application to input the name of an MFS control block to an IMS application program or to retrieve the name of an MFS control block provided by an IMS application program.

On input, typically the value of the mapName property is the name of an MFS Message Output Descriptor, or ″MOD″. The MOD name will be provided to the IMS application program in the I/O PCB.

On output, the value of the mapName property is the name of an MFS Message Output Descriptor, or ″MOD″. This is the MOD name that the IMS application program specified when inserting the transaction output message to the I/O PCB.

Note: The mapName field should not be used by Java applications that use an enterprise service whose input and output messages are generated by WebSphere Studio MFS support.

purgeAsyncOutput

This is an input property. This property determines whether or not IMS Connect purges undelivered output.

This property is only valid for interactions on shareable persistent socket connections that use IMS interaction verb SYNC_SEND_RECEIVE. It is not valid for any interactions on dedicated persistent socket connections. It applies to commit mode 0 interactions. It does not apply to commit mode 1 interactions. However, if a commit mode 1 interaction executes a program-to-program switch, the spawned program will run commit mode 0 and therefore the property will apply.

If the purgeAsyncOutput property is not specified on a SYNC_SEND_RECEIVE interaction on a shareable persistent socket connection, the default is TRUE and the following output messages are purged:

- Undelivered output message inserted to the I/O PCB by the primary IMS application program.
- Output messages inserted to the I/O PCB by secondary IMS application programs invoked by a program to program switch.

reRoute

This is an input property.

This property is only valid for interactions on shareable persistent socket connections that use IMS interaction verb SYNC_SEND_RECEIVE. It is not valid for any interactions on dedicated persistent socket connections. It applies to commit mode 0 interactions. It does not apply to commit mode 1 interactions. However, if a commit mode 1 interaction executes a program-to-program switch, the spawned program will run commit mode 0 and therefore the property will apply. This property determines if undelivered output is to be rerouted to a named destination specified in the reRouteName field. If reRoute is TRUE, the asynchronous output is not queued to the TPIPE of the generated clientID. Instead, the asynchronous output is queued to the destination specified in the reRouteName field. The default value for reRoute is FALSE.

If both reRoute and purgeAsyncOutput are set to TRUE, an exception is thrown.

reRouteName

This property provides the name of the destination to which asynchronous output is queued. If reRoute is TRUE, this property provides the named destination. If reRoute is FALSE, the reRouteName property is ignored.

If the reRoute property is set to TRUE, and no reRouteName is provided, the value for the reRouteName property is:

1. The value specified in the IMS Connect configuration file.
2. If no value is specified in the IMS Connect configuration file, the value ″HWS$DEF″ is used.

Valid values for the reRouteName property:

- Must be a string of 1 to 8 alphanumeric (A-Z, 0-9) or special (@,#,$) characters.
- Must not start with the character string, “HWS”.
- Must not be an IMS Connect port number.
- If lowercase letters are provided, the letters will be changed to uppercase.

The property, reRouteName, is only valid for SYNC_SEND_RECEIVE interactions on shareable persistent socket connections. It is not valid for any interactions on dedicated persistent socket connections.

required

Leave this field empty.

Chapter 6. Security

The topics in this section describe security issues for the IMS resource adapter. The topics included are:

IMS resource adapter security

Information in an Enterprise Information System (EIS) such as IMS must be protected from unauthorized access. The J2EE Connector Architecture (J2C) specifies that the application server and the EIS must collaborate to ensure that only authenticated users are able to access an EIS. The J2C security architecture extends the end-to-end security model for J2EE-based applications to include integration with EISs.

EIS sign-on

The J2C security architecture supports a user ID and password authentication mechanism specific to an EIS. For more information, see Java 2 Connector security in the WebSphere Application Server documentation.

The user ID and password for the target EIS is supplied either by the application component (component-managed sign-on) or by the application server (container-managed sign-on).

For IMS Connector for Java, IMS is the target EIS. The security information is passed to the IMS resource adapter, which then passes it to IMS Connect. IMS Connect uses this information to perform user authentication and passes it on to IMS OTMA which also uses this information to verify authorization to access IMS.

In a typical environment, the IMS resource adapter passes on the security information (user ID, password, and optional group name) that it receives to IMS Connect in an IMS OTMA message. Depending on its security configuration, IMS Connect may then call the host's Security Authorization Facility (SAF).

- For WebSphere Application Server on distributed platforms or z/OS with TCP/IP, using either component-managed or container-managed sign-on:
  - If RACF=Y is set in the IMS Connect configuration member or if the IMS Connect command `SETRACF ON` has been issued, IMS Connect calls the SAF to perform authentication using the user ID and password passed by IMS Connector for Java in the OTMA message. If authentication succeeds, the user ID, groupname, and UTOKEN returned from the IMS Connect call to the SAF are passed to IMS OTMA for use in verifying authorization to access IMS.
  - IF RACF=N is set in the IMS Connect configuration member or if the IMS Connect command `SETRACF OFF` has been issued, IMS Connect does not call the SAF. However, the user ID and groupname are still passed to IMS OTMA for use in verifying authorization to access IMS.
- For WebSphere Application Server on z/OS with Local Option, using either component-managed or container-managed sign-on:
  - Regardless of the RACF® setting in the IMS Connect configuration member or in the `SETRACF` command, IMS Connect does not call the SAF, because authentication has already been performed by WebSphere Application Server for z/OS. The UTOKEN generated when WebSphere Application Server for z/OS calls RACF is passed to IMS for use in verifying authorization to access IMS.
  - WebSphere Application Server for z/OS can be configured to use the user identity associated with the thread of execution to authenticate a user. The application server creates and passes the UTOKEN representing the user identity to the IMS resource adapter. The IMS resource adapter then passes the token to IMS Connect for sign-on to IMS. For information about the RunAs Identity support in WAS, consult the security documentation for WebSphere Application Server z/OS.

The level of authorization checking performed by IMS is controlled by the IMS command, /SECURE OTMA. See the *IMS OTMA Guide and Reference* for more information about this command.

Java2 Security Manager

The IMS resource adapter works with the WebSphere Application Server Java2 Security Manager. Components such as resource adapters must be authorized to perform protected tasks, such as making socket calls. The IMS resource adapter is already authorized to perform these tasks. No action is required by the application component.

See the Managing secured applications in the WebSphere Application Server documentation for more information about the Java2 Security Manager.

Component-managed EIS sign-on

When you specify <res-auth>Application</res-auth> in the deployment descriptor of your application, component-managed EIS sign-on is used.

Your application (the component) should provide the security information (user ID, password, and optional group name) used for EIS sign-on:

- If your application uses the J2EE Connector Architecture Common Client Interface (CCI), it performs component-managed sign-on by first populating an IMSConnectionSpec object with the security information. Then, when the application establishes a connection to IMS, it passes the IMSConnectionSpec object as a parameter of the IMSConnectionFactory.getConnection method. The IMS resource adapter uses this security information for the sign-on to IMS.
- If your application is an application built by Rational Application Developer, the security information is passed as application input data. To pass the security information as input data you must expose the properties, userName, password, and groupName of IMSConnectionSpec. For specific information about how to expose the IMSConnectionSpec properties user ID, password, and group name as application input data for the IMS resource adapter, see "Exposing InteractionSpec and ConnectionSpec properties for input as data" on page 13.

If your application does not use one of the above methods to provide security information, WebSphere Application Server will obtain the security information from the J2C connection factory's custom properties. **Note:** If you specified a component-managed JAAS Authentication alias while setting up your connection factory, the user ID and password in the alias will override the userName and password values in the connection factory custom properties during the start-up of the WebSphere Application Server.

Configuring component-managed EIS sign-on

In most cases, when you create a J2EE application using the wizards of Rational Application Developer, the default EIS sign-on is component-managed. The component-managed configuration setting is reflected by the <res-auth>Application</res-auth> directive of the resource reference used by your application.

The following steps explain how to verify or change this setting for a Dynamic Web Project.

1. Set the <res-auth> directive to Application
   a. In the J2EE perspective, of the Project Explorer view, expand **Dynamic Web Project > PhoneBookWeb**.
   b. Right-click **Deployment Descriptor: PhoneBookWeb** and select **Open With > Deployment Descriptor Editor**.
   c. In the Web Deployment Descriptor view, click the **References** tab and select the J2C Connection factory reference for your Web application. For example, *imsCFacRef*.

d. Select **Application**, if it is not already selected, in the **Authentication** field, which maps to the `<res-auth>` directive.

e. When you close the Web Deployment Descriptor Editor and click **Yes** to save your changes; the following code is added to the deployment descriptor of your Web application:

```
<res-auth>Application</res-auth>
```

2. Typically, component-managed sign-on does not require further configuration because the security information is provided by the application in the IMSConnectionSpec object. However, if your application does not provide an IMSConnectionSpec object, or the user ID is not specified in the IMSConnectionSpec object that is provided, the IMS resource adapter will obtain default security values from the connection factory used by your application.

   The default security values for a connection factory can be provided in two ways:

   a. When you use a component-managed authentication alias.

      - To use a component-managed authentication alias, you must define a JAAS authentication alias.
        1) In the Servers view, right-click the server and select **Run administrative console**.
        2) Expand Resources and select **Resource Adapters**.
        3) Click the resource adapter you want to modify.
        4) Under Additional Properties, click **J2C Connection factories**.
        5) Under Related Items, click **J2EE Connector Architecture (J2C) authentication data entries**.
        6) Above the list of aliases, click **New**.
        7) Enter an alias name, your user ID, password, and optional description. Select **OK**.
      - Select the JAAS authentication alias for the Component-managed authentication alias property of the J2C Connection Factory used by your application. You can do this when you first create the connection factory or later by editing the connection factory. To edit the connection factory:
        1) In the Administrative Console for the server, navigate to the connection factory that you wish to modify by selecting **Resource Adapters >** *server_name* **> J2C connection factories >** *connection_factory_name*.
        2) In the Component-managed authentication alias drop down list, select the JAAS authentication alias to be used for component-managed authentication by applications using that connection factory.
        3) Select **OK**.

        The user ID and password associated with the component-managed authentication alias will be used to set (over override if applicable) the default values in the custom properties of the associated connection factory during application server startup.

   b. When you create a connection factory.

      - If you do not assign a valid JAAS authentication alias to the component-managed authentication alias field of your J2C connection factory, you can assign values for the userName, password, and groupName fields on the J2C options page of your J2C connection factory.
      - For instructions on creating a connection factory, see Connection Properties. Using a component-managed authentication alias is preferred over specifying values in the custom properties of your J2C connection factory because the component-managed authentication alias provides greater security for the user ID and password.

Note: The process for configuring component-managed sign-on in a standalone WebSphere Application Server is the same as the process for a WebSphere Application Server in a unit test environment.

Container-managed EIS sign-on

When <res-auth>Container</res-auth> is specified in the deployment descriptor of the application, container-managed EIS sign-on will be used. When container-managed sign-on is used, your application does not programmatically provide the security information. Instead, the application server (the container) provides the security information (user ID and password). One way to accomplish this when using DefaultPrincipalMapping, is to provide values for the user ID and password to be used by the application server as follows:

- Define a JAAS Authentication alias, associating the user ID and password you wish to use for EIS sign-on with the alias
- Associate this alias with the J2C connection factory used by your application

For TCP/ IP, the application server passes the security information in the alias to the IMS resource adapter. The IMS resource adapter passes the security information to IMS Connect for authentication. IMS Connect authenticates the user and passes the security information for sign-on to IMS. If IMS Connect cannot authenticate the user, a security failure is returned to the IMS resource adapter which, in turn, passes an exception back to the application.

For Local Option, a z/OS-only feature in which both the server and WebSphere Application Server are running in the same MVS image, the application server authenticates the user based on the security information defined in the container-managed alias. The application server creates and passes a UTOKEN representing the authenticated user to the IMS resource adapter. The IMS resource adapter then passes the UTOKEN to IMS Connect which in turn passes it on to IMS OTMA for use in signing on to IMS.

Alternatively, when using Local Option communications, you can specify in the application server configuration that the user identity associated with the current thread of execution is to be used by the application server when performing user authentication. In this case, you do not specify a JAAS container-managed authentication alias in the J2C connection factory used by your application. This option is only available if you are using Local Option communications.

Note: When using container-managed sign-on, if your application does pass security information to the IMS resource adapter using the userName, password or groupName properties of IMSConnectionSpec, it is ignored. However, if you pass other information in the IMSConnectionSpec object, such as clientID used with commit mode 0 interactions, this information will be used by the IMS resource adapter.

Configuring container-managed EIS sign-on

Although the method for configuring container-managed EIS sign-on is deprecated in WebSphere Application Server Version 6, this topic uses a Dynamic Web Project to illustrate how to configure a component for container-managed EIS sign-on.

This configuration setting is reflected by the <res-auth>Container</res-auth> directive of the resource reference used by your application.

1. Set the <res-auth> directive to Container.
   a. In the J2EE perspective of the Project Explorer view, expand **Dynamic Web Projects > PhoneBookWeb**.
   b. Right-click **Deployment Descriptor: PhoneBookWeb** and select **Open With > Deployment Descriptor Editor**.
   c. In the Web Deployment Desccriptor view, click the **References** tab and select the J2C Connection factory reference for your Web application. For example, select *imsCFacRef*..
   d. In the **Authentication** field, select **Container** from the drop-down list, if it is not already selected.
   e. When you close the Web Deployment Descriptor Editor and click **Yes** to save your changes. The following code is added to the deployment descriptor of your Web application:

```
<res-auth>Container</res-auth>
```

2. Specify a method of providing the user ID and password that you want the application server. To use a JAAS authentication alias to provide the user ID and password that you can use for EIS sign-on, complete the following steps:
   a. In the Servers view, right-click the server and select **Run administrative console.**
   b. Expand Resources and select **Resource Adapters**.
   c. Select the resource adapter you want to modify.
   d. Under Additional Properties, click **J2C connection factories**.
   e. Under Related Items, click **J2EE Connector Architecture (J2C) authentication data entries**.
   f. Above the list of aliases, click **New**.
   g. Enter an alias name, your user ID, password, and optional description. Select **OK**.
3. Select the JAAS authentication alias for the Container-managed authentication alias property of the J2C connection factory used by your application. You can do this when you first create the connection factory or later by editing the connection factory. To edit the connection factory:
   a. In the Administrative Console for the server you selected, navigate to the connection factory that you wish to modify. For example, **Resource adapters >** *server_name* **> J2C connection factories >***connection_factory_name*.
   b. In the Container-managed authentication alias drop down list, select the JAAS authentication alias to be used for the container-managed authentication by applications using that connection factory.
   c. Select **OK**.

For information about the properties of a connection factory, see Connection Properties.

Note: The process for configuring container-managed sign-on in a standalone WebSphere Application Server is the same as the process for a WebSphere Application Server in a unit test environment.

Overview of secure socket layer (SSL)

With the evolution of e-business, data security has become very important for Internet users. The Secure Socket Layer (SSL) protocol ensures that the transfer of sensitive information over the Internet is secure. SSL protects information from:

- Internet eavesdropping
- Data theft
- Traffic analysis
- Data modification
- Trojan horse browser /server

One way IMS Connector for Java communicates with IMS Connect is through TCP/ IP sockets. If IMS Connector for Java uses TCP/ IP, SSL can be used to secure the TCP/ IP communication between the two entities. The SSL support provided by IMS Connector for Java, along with the support provided by IMS Connect, uses a combination of public and private keys along with symmetric key encryption schemes to achieve client and server authentication, data confidentiality, and integrity. SSL rests on top of TCP/ IP communication protocol and allows an SSL-enabled server to authenticate itself to an SSL-enabled client and vice versa. For an SSL connection between IMS Connector for Java and IMS Connect, IMS Connector for Java is considered to be the client and IMS Connect is considered to be the server. Once authentication is complete, the server and client can establish an encrypted connection that also preserves the integrity of the data.

For SSL support when running in a WebSphere environment, IMS Connector for Java uses the IBM® implementation of Java Secure Socket Extension (IBM JSSE). The SSL library is included in WebSphere Studio Application Developer Integration Edition and in WebSphere Application Server.

SSL concepts

Certificate

A digital certificate is a digital document that validates the identity of the certificate's owner. A digital certificate contains information about the individual, such as their name, company, and public key. The certificate is signed with a digital signature by the Certificate Authority (CA), which is a trustworthy authority.

Certificate authority

A Certificate Authority (CA) is a trusted party that creates and issues digital certificates to users and systems. The CA, as a valid credential, establishes the foundation of trust in the certificates.

Certificate management

Certificates and private keys are stored in files called keystores. A keystore is a database of key material. Keystore information can be grouped into two categories: key entries and trusted certificate entries. The two entries can be stored in the same keystore or separately in a keystore and trustore for security purposes. Keystores and truststores are used by both the SSL client, IMS Connector for Java, and the SSL server, IMS Connect.

Keystore
: A keystore holds key entries, such as the private key of the user. For example, the client IMS Connector for Java.

Truststore
: A truststore is a keystore that holds only certificates that the user trusts. An entry should be added to a truststore only if the user makes a decision to trust that entity. An example of an IMS Connector for Java (client) truststore entry would be the certificate of the target server, IMS Connect.

For convenience, IMS Connector for Java allows the user to store key entries and trusted certificate entries in either the keystore or the truststore. The user may still choose to store them separately. IMS Connector for Java supports only X.509 certificates and the "JKS" keystore type on distributed platforms (which include zLinux) and the "JKS" keystore type or RACF keyrings on OS/390 and z/OS.

SSL process

The SSL protocol consists of server authentication, client authentication (optional but strongly recommended) followed by an encrypted conversation. The following scenario steps through the SSL process.

Server authentication

SSL server authentication allows a client to confirm a server's identity. SSL-enabled client software uses standard techniques of public-key cryptography to ensure that a server's certificate and public ID is valid and that the certificate and ID was issued from one of the client's list of trusted certificate authorities (CA).

Client authentication

SSL client authentication allows a server to confirm a client's identity. Using the same techniques used for server authentication, SSL-enabled server software verifies that a client's certificate and public ID is valid and that the certificate and ID was issued by one of the server's list of trusted certificate authorities (CA).

SSL handshake

Both the client, IMS Connector for Java, and the server, IMS Connect, store their certificates and private keys in keystores. The actual SSL session between IMS Connector for Java and IMS Connect is established by following a handshake sequence between client and server. The sequence will vary depending on whether the server is configured to provide a server certificate or to request a client certificate, and which cipher suites are being used. A cipher is an encryption algorithm. The SSL protocol determines how the client and server negotiate the cipher suites to authenticate one another, to transmit certificates, and to establish session keys. Some of the algorithms used in cipher suites include:

- DES - Data Encryption Standard
- DSA - Digital Signature Algorithm
- KEA - Key Exchange Algorithm
- MD5 - Message Digest algorithm
- RC2 and RC4 - Rivest encryption ciphers
- RSA - A public key algorithm for both encryption and authentication
- RSA key exchange - A key-exchange for SSL based on the RSA algorithm
- SHA-1 - Secure Hash Algorithm
- SKIPJACK - A classified symmetric-key algorithm implemented in FORTEZZA-compliant hardware
- Triple-DES - DES applied three times.

SSL 2.0 and SSL 3.0 protocols support overlapping sets of cipher suites. Administrators can enable or disable any of the supported cipher suites for both clients and servers. When a particular client and server exchange information during the SSL handshake, the client and server identify the strongest enabled cipher suites that they have in common and use one of them for the SSL session.

Transport Layer Security, Version 1 (TLS V1) is the successor to SSL 3.0 protocol. IMS Connector for Java **only** supports TLS V1. There are no backward compatibility issues.

Using secure socket layer (SSL) support

The following table provides a high level description of how IMS Connector for Java and IMS Connect SSL support is set up and configured. Follow the steps in the order outlined below:

SSL Client (IMS Connector for Java)	SSL Server (IMS Connect)
	1. Decide if client authentication is required. **Note:** It is strongly recommended that you do use client authentication to protect against unauthorized access to your IMS Connect. If client authentication is not required, skip to Step 5.
2. If client authentication is required, obtain signed certificates and private key.	
3. If client authentication is required, create a keystore and insert the client's private key and certificate. For more detail, see the description below.	
	4. If client authentication is required, insert the client's public key certificate into the keyring. See *IMS Connect User's Guide* (SC27-0946-03) for more information.
5. Create a truststore (another optional keystore) and insert the Server's public key certificate. Alternatively, you would insert the Server's public key certificate into the client's keystore if trusted and non-trusted certificates are stored in the same keystore.	

SSL Client (IMS Connector for Java)	SSL Server (IMS Connect)
	6. Decide which IMS Connect SSL port to use. Set up the IMS Connect and SSL Configuration members with the appropriate values. For more information about setting up these configuration members, see *IMS Connect User's Guide* (SC27-0946).
7. Set up the connection factory with the appropriate SSL parameters, including the port number from step 6.For more detail, see the description below.	
8. Bind the application to the SSL connection factory.	

Creating the keystore or truststore for the client

For the client and server to authenticate one another, you must provide a JKS keystore or RACF keyring with valid X.509 certificates at both the client and server ends. If client authentication by the server is not required, it is not necessary to create the client certificate and add it to the server's keyring. There are several tools available for managing the keystore. To provide a JKS keystore at both the client and server ends, you must perform the following steps:

- To set up the client, IMS Connector for Java, create a certificate and have it signed by a Certificate Authority (for example, VeriSign), or create your own Certificate Authority (CA) using software such as OpenSSL to sign your own (self-signed) certificate.
- To create a keystore, use a key management tool such as Ikeyman or Keytool. After the keystore is created, import the client certificate (if one is available) into the keystore.
- To create a truststore, create another keystore and import the server's (IMS Connect's) certificate. **Note**: If you want to create only one keystore, import the server's certificate into the same keystore used to store the client's certificates.

SSL configuration

A secure SSL connection between a Java client application and IMS Connect is created by ensuring that the connection factory used by the Java client application has the appropriate values for its SSL properties. See Connection properties for a description of the SSL property values.

There are two ways to set up SSL properties:

1. If you are running your Java client application in the Unit Test Environment of WebSphere Studio Application Developer Integration Edition, you use the tooling in WebSphere Studio Application Developer Integration Edition. WebSphere Studio Application Developer Integration Edition maps or binds the connection factory resource reference in the Java client application, which is installed on WebSphere Application Server, to the SSL-configured connection factory by providing the JNDI name of the connection factory.
2. If you are running your Java application in WebSphere Application Server, you can configure a connection factory so that it will create SSL connections by setting appropriate values for the SSL-related properties in the Custom Properties of a of a J2C Connection Factory in WebSphere Application Server. To set a connection factory's Custom Properties, navigate to **Resources -> Resource Adapters -> myIMSResourceAdapter -> J2C Connection Factories -> myJ2CConnectionFactory -> Custom Properties** in the WebSphere Application Server administrative console.

   The following figure displays a J2C Connection Factory Custom Properties property sheet:

HostName	myHostNm	TCP/IP host name of the target IMS Connect	false
PortNumber	0	Target TCP/IP port number of IMS Connect	false
DataStoreName	myDStrNm	Name of the target IMS datastore	false
SSLEnabled	FALSE	Indicates if SSL is enabled for this connection factory	false
SSLEncryptionType	Weak	The type of cipher suite to be used for encryption	false
SSLKeyStoreName		Name (full path) of SSL keystore for client certificates/private keys	false
SSLKeyStorePassword		Password of SSL keystore for client certificates/private keys	false
SSLTrustStoreName		Name (full path) of SSL keystore for trusted certificates	false
SSLTrustStorePassword		Password of SSL keystore for trusted certificates	false
CM0Dedicated	FALSE	Indicates if sockets are dedicated to specific CM0 clients	false
UserName		Default name of the user to be authorized	false
Password		Default password of the user	false

Note: Informational messages and warnings can be found in the **trace.log** file generated by WebSphere Application Server.

At runtime, when the Java client application executes an interaction with IMS, the interaction flows on a secure (SSL) connection IMS Connector for Java to IMS Connect. The following steps are transparent to the Java client application. The IMS resource adapter interacts with IMS Connect using the SSL protocol as follows:

- The SSL client, IMS Connector for Java initiates a connection by sending a client hello. The server, IMS Connect, replies with a server hello and its certificate containing its public key.
- If the server does not require client authentication, the client authenticates the server's certificate using the server's public key from its certificate. If authentication is successful, the SSL handshake is completed. A session key has been established at both ends.
- If the server does require client authentication, the client authenticates the server's certificate using the server's public key from its certificate. If this authentication is successful, a client certificate is sent from the client's keystore. If this certificate is authenticated successfully by the server, the SSL handshake is completed. A session key has been established at both ends.
- The client and server are then ready to send and receive encrypted data.

It is important to note that, when running applications in a managed environment (as is strongly recommended when using SSL connections,) IMS Connector for Java uses only persistent socket connections to communicate with IMS Connect. When the WebSphere Application Server Connection Manager is used, connections can be serially reused by other client applications. The connection manager creates connections if necessary, and provides them to the applications as needed. When an application is finished using a connection, the connection manager returns that connection to the free pool making it available for reuse by any other application requiring that type of connection. However, client and server authentication only occurs once for each socket during the handshake that takes place when that socket is

first created and initialized as an SSL socket. When a socket is reused, the SSL client, IMS Connector for Java, and server, IMS Connect, do not change. Consequently, there is no reason to re-authenticate the client and server (go through the handshake process again) when a socket is reused. Note that this is consistent with the fact that the clientID which identifies a socket remains the same each time a socket is reused.

Chapter 7. Commit mode processing

The topics in this section describe some of the different processing models that a Java client can use with the IMS resource adapter. The topics included are:

Overview of commit mode processing

Commit mode refers to the type of commit processing performed by IMS. The Java client specifies the commit mode protocol to be used when it submits a transaction request to IMS. There are two types of commit mode processing supported by IMS Connect and IMS: commit mode 0 (commit-then-send) where IMS commits the IMS database changes and then sends the output to the client and commit mode 1 (send-then-commit) where IMS sends the output to the client and then commits the database changes.

Associated with the commit mode protocols, IMS Connect and IMS also support three synchronization levels (synch levels): NONE, CONFIRM, and SYNCPT. All three synch levels can be used with commit mode 1. Only CONFIRM can be used with commit mode 0. However, IMS Connector for Java does not currently support commit mode 1, synch level CONFIRM.

Currently, the synchronization level is not set by the Java client. IMS Connector for Java automatically provides the synchronization level when communicating with IMS Connect.

IMS Connector for Java supports the following combinations:

- Commit mode 1 with synch level NONE

  This combination is used for non-transactional interactions. For non-conversational applications, use the SYNC_SEND_RECEIVE interaction. For conversational applications, use SYNC_SEND_RECEIVE or optionally, SYNC_END_CONVERSATION interaction.
- Commit mode 1 with synch level SYNCPT

  This combination is used by IMS Connector for Java when participating in two-phase commit processing with IMS. For more information, see Global transaction support with two-phase commit.
- Commit mode 0 with synch level CONFIRM

  This combination is used by IMS Connector for Java for non-transactional SYNC_SEND_RECEIVE, SYNC_SEND, SYNC_RECEIVE_ASYNCOUTPUT, SYNC_RECEIVE_ASYNCOUTPUT_SINGLE_NOWAIT and SYNC_RECEIVE_ASYNCOUTPUT_SINGLE_WAIT interactions.

  Note: Commit mode 0 is only supported for non-conversational applications running on TCP/IP connections.

The synchronization level is not set by the Java client. IMS Connector for Java automatically provides the synchronization level when communicating with IMS Connect.

If the Java client submits a transaction request with commit mode 1 synch level NONE, IMS Connector for Java passes the request through IMS Connect to IMS. IMS processes this transaction and attempts to send the output message to the Java client. The Java client may receive the output message from the transaction or may receive an exception. In either case, IMS will have already committed the changes to the database and discarded the output message of the IMS transaction.

Similarly, if the Java client sends a transaction with commit mode 0 synch level CONFIRM, the Java client may receive the output message from the transaction or may receive an exception. However, if the Java client receives an exception when commit mode 0 is used, the output may or may not be queued for later retrieval. Whether or not the output message that was not delivered to a Java client will be queued depends on the type of socket connection the Java client uses for the commit mode 0 interaction.

The type of exception also determines whether or not an output message is available for retrieval. For example, if the Java client receives an IMSDFSMessageException indicating that the transaction is stopped, the application was not run; therefore, there is no output message available for retrieval. However, if the transaction runs but the executionTimeout value expires before the output message is returned to IMS Connect, the Java client will receive an EISSystemException that an execution timeout has occurred. In this case, the output message will be queued to the appropriate IMS OTMA Asynchronous Output Queue or TPIPE for later retrieval.

Note: In IMS/OTMA terminology, a transaction pipe (TPIPE) is a logical connection between a client (IMS Connect) and the server (IMS/OTMA). For commit mode 0 interactions, the TPIPE is identified by the clientID used for the interaction. Each clientID used for a commit mode 0 transaction will have its own TPIPE. For commit mode 1 interactions, the TPIPE is identified by the IMS Connect port number used for the interaction. Therefore, each port will have a TPIPE which will be used for all clients running commit mode 1 interactions on that port.

Regardless of whether your Java client is running an IMS transaction with commit mode 1 or commit mode 0, the Java client specifies a value for the interactionVerb property of IMSInteractionSpec. If a commit mode 0 interaction is specified, the Java client may also have to provide a value for the clientID property of IMSConnectionSpec. clientID is a property of IMSConnectionSpec and identifies the IMS OTMA Asynchronous Output Queue or TPIPE where the recoverable output messages are placed. Whether or not a Java client provides a clientID for a commit mode 0 interaction depends on the type of socket connection being used by the Java client.

To retrieve output messages from a TPIPE, the Java client submits a request in which it specifies one of the values SYNC_RECEIVE_ASYNCOUTPUT, SYNC_RECEIVE_ASYNCOUTPUT_SINGLE_NOWAIT, or SYNC_RECEIVE_ASYNCOUTPUT_SINGLE_WAIT for the interactionVerb property of IMSInteractionSpec and a value for the clientID property of IMSConnectionSpec. For more information about asynchronous output support, see *Chapter 9: Protocols* in *IMS Connect Guide and Reference*.

In general, the SYNC_RECEIVE_ASYNCOUTPUT, SYNC_RECEIVE_ASYNCOUTPUT_SINGLE_NOWAIT, or SYNC_RECEIVE_ASYNCOUTPUT_SINGLE_WAIT interactions can be used to retrieve output messages queued for any clientID, regardless of how those messages were queued to the associated clientID - either as a result of a failed commit mode 0 transaction or from an IMS application that issued an insert to an ALTPCB (Alternate Program Communication Block). In the case of retrieving an output message from a failed commit mode 0 transaction, the clientID provided in the IMSConnectionSpec for retrieval request must match the clientID that was specified on the failed commit mode 0 transaction.

If there is nothing in the OTMA Asynchronous Output Queue for that particular clientID, you will receive an execution timeout exception. The timeout exception can mean either that there are no messages in the queue or that the timeout value did not provide enough time for IMS Connect to retrieve the message from the queue. For both SYNC_RECEIVE_ASYNCOUTPUT, SYNC_RECEIVE_ASYNCOUTPUT_SINGLE_NOWAIT, or SYNC_RECEIVE_ASYNCOUTPUT_SINGLE_WAIT, as well as SYNC_SEND_RECEIVE interactions, executionTimeout is the length of time IMS Connect will wait for a response from IMS. If you do not specify an execution timeout value for a retrieval request, the default execution timeout value will be used. The default timeout value is the IMS Connect configuration member TIMEOUT value. The user may need to experiment with the execution timeout value, to ensure that output messages are returned for all types of interactions.

Commit mode processing and socket connections

All socket connections created by the IMS resource adapter are persistent. In other words, the same socket connection between IMS Connector for Java and IMS Connect can be serially reused for multiple interactions with IMS Connect. The socket connection will not be closed and reopened between interactions. There are two types of persistent sockets; shareable and dedicated.

Shareable Persistent Socket

The shareable persistent socket can be shared (serially reused) by multiple applications executing either commit mode 1 or commit mode 0 interactions. For an application executing a commit mode 0 interaction on a shareable persistent socket, the IMS resource adapter automatically generates a clientID with the prefix ″HWS″. This clientID represents and identifies the socket connection as well as the associated OTMA TPIPE. For this type of socket, only clientIDs generated by the IMS resource adapter are allowed. A user-specified clientID is not allowed with shareable persistent socket support.

Note: IMS application programs that insert messages to an alternate PCB must not use names beginning with ″HWS″ for the alternate PCBs.

Any output message that cannot be delivered to a Java client executing a commit mode 0 interaction on a shareable persistent socket can be queued for later retrieval. Also, any commit mode 1 or commit mode 0 interaction on a shareable persistent socket that spawns a program-to-program switch which invokes another commit mode 0 interaction resulting in secondary output, can be requeued for later retrieval. SYNC_RECEIVE_ASYNCOUTPUT, SYNC_RECEIVE_ASYNCOUTPUT_SINGLE_NOWAIT, and SYNC_RECEIVE_ASYNCOUTPUT_SINGLE_WAIT interactions are supported on shareable persistent sockets. To retrieve undelivered output messages that are queued in the IMS OTMA Asynchronous Hold Queue or TPIPE, the interaction verbs must be invoked within the same client application, because the same generated client ID that identifies the shareable socket connection and the associated OTMA TPIPE must be used.

On shareable persistent sockets, the undelivered output messages can be handled in more than one way. One way is to purge the undelivered output. To purge undelivered output messages, you must ensure the IMSInteractionSpec property purgeAsyncOutput is TRUE. This input property determines if IMS Connect purges the undelivered I/O PCB output. The purgeAsyncOutput property is only valid with the SYNC_SEND_RECEIVE interaction verb. If the property is not specified on SYNC_SEND_RECEIVE, the default is TRUE.

Another option of handling undelivered output messages on shareable persistent sockets is rerouting the messages to another destination. You can reroute the undelivered output message to a different destination by setting the IMSInteractionSpec property, reRoute, to TRUE. This property is only valid for the SYNC_SEND_RECEIVE interaction verb. If reRoute is set to TRUE, the undelivered output message is queued to a named destination provided by the client application, which is specified on the reRouteName IMSInteractionSpec property. If the reRoute property is set to TRUE and no reRouteName is provided, the value of the reRouteName property is the value specified in the IMS Connect configuration file. If no value is specified in the IMS Connect configuration file, the default value HWS$DEF is used.

Shareable persistent socket connections are created by an IMS Connection Factory with values for at least the following custom properties:

- Host name = TCP/IP host name of machine running IMS Connect
- Port number = associated port number
- Datastore name = name of target IMS
- CM0Dedicated = FALSE

FALSE is the default value for the endCM0Dedicated property and ensures that the connection factory will create shareable persistent socket connections.

Dedicated persistent socket

A dedicated persistent socket is used for Java applications executing commit mode 0 interactions only. It can be shared (serially reused) by multiple applications with the same user-specified clientID. For this type of socket, only interactions with user-specified clientIDs are allowed. A valid user-specified clientID:

- Must be a string of 1 to 8 alphanumeric (A-Z, 0-9) or special (@,#,$) characters.
- Must not start with the character string, ″HWS″.
- Must not be an IMS Connect port number.
- If lowercase letters are provided, the letters will be changed to uppercase

A dedicated persistent socket means the socket connection is assigned to a specific clientID and will remain dedicated to that particular clientID until it is disconnected. SYNC_SEND_RECEIVE, SYNC_SEND, SYNC_RECEIVE_ASYNCOUTPUT, SYNC_RECEIVE_ASYNCOUTPUT_SINGLE_NOWAIT, and SYNC_RECEIVE_ASYNCOUTPUT_SINGLE_WAIT interactions are supported on dedicated persistent sockets.

SYNC_RECEIVE_ASYNCOUTPUT, SYNC_RECEIVE_ASYNCOUTPUT_SINGLE_NOWAIT, and SYNC_RECEIVE_ASYNCOUTPUT_SINGLE_WAIT interactions on dedicated persistent sockets enable client applications to retrieve messages that were placed on an IMS OTMA Asynchronous Output Queue as a result of a failed commit mode 0 interaction, from an IMS application that issued an insert to an ALTPCB (Alternate Program Communication Block), or from the reroute of the output from a transaction that was executed on a shareable connection factory. To retrieve the messages, the client application must provide the clientID, which represents the TPIPE that has asynchronous output messages queued. Interactions on dedicated persistent sockets that have undelivered output messages cannot be rerouted or purged.

Dedicated persistent socket connections are created by an IMS Connection Factory with values for at least the following custom properties:

- Host name = TCP/IP host name of machine running IMS Connect
- Port number = associated port number
- Datastore name = name of target IMS
- CM0Dedicated = TRUE

A value of TRUE for the endCM0Dedicated property ensures that the connection factory will create dedicated persistent socket connections.

Note: If you have more than one connection factory configured to create dedicated persistent sockets to the same IMS Connect instance, only one connection factory can dedicate a socket to a particular clientID at one time. For example, if the first connection factory successfully creates a socket connection dedicated to clientID, CLIENT01; the second connection factory will receive the following exception if it tries to create a socket connection dedicated to CLIENT01 while the socket connection created by the first connection factory is still connected to IMS Connect:

```
javax.resource.spi.EISSystemException: ICO0001E:
com.ibm.connector2.ims.ico.IMSTCPIPManagedConnection@23766050.processOutputOTMAMsg
(byte [], InteractionSpec,Record) error. IMS Connect returned error: RETCODE=[8],
REASONCODE=[DUPECLNT].
Duplicate client ID was used;  the client ID is currently in use.
```

Releasing Persistent Sockets

A TCP/IP connection between IMS Connector for Java and IMS Connect is persistent; in other words it remains open as long as IMS Connector for Java or IMS Connect does not disconnect it due to an error. This is the case for both a shareable persistent socket connection and a dedicated persistent socket connection. However, in the case of a dedicated persistent socket connection, the socket connection can only be used by interactions that have the same clientID that was used to establish the connection. The number of socket connections will increase as new clientIDs are used for interactions on dedicated persistent socket connections.

If you have the Max connections property set to a non-zero value and you also have a non-zero value for the Connection timeout property, when the MaxConnections is reached and all the connections are in use,

the application will get a ConnectionWaitTimeoutException after the seconds specified in Connection timeout have elapsed. This is standard behavior for WebSphere Application Server. The ConnectionWaitTimeoutException applies to both dedicated persistent sockets and shareable persistent sockets.

However, if MaxConnections has been reached and one of the persistent socket connections is currently not in use, then WebSphere Application Server will disconnect that socket in order to respond to the request to create a new persistent socket connection. This also is standard behavior for the WebSphere Application Server and applies to both dedicated and shareable persistent sockets.

SYNC_SEND programming model

If your Java client application issues a SYNC_SEND interaction, the IMS resource adapter sends the request to IMS through IMS Connect and does not expect a response from IMS. Because the IMS resource adapter performs a ″send only″ interaction with IMS, a SYNC_SEND interaction is typically used with a non-response mode transaction.

To use a SYNC_SEND interation to run a transaction, your application must provide a value of SYNC_SEND for the interactionVerb property and a value of 0 for the commitMode property of the IMSInteractionSpec object used by the execute method. SYNC_SEND interaction processing varies depending on the type of pesistent socket used (shareable or dedicated) and the type of IMS transaction that is run.

Note: IMSInteractionSpec properties purgeAsycOutput, reRoute and reRouteName do not apply to SYNC_SEND interactions and are ignored by IMS Connector for Java.

Shareable persistent socket processing model

The following scenarios describe a SYNC_SEND interaction on a shareable persistent socket connection for different type of transactions.

- Non-response mode transaction

  An IMS application program associated with a transaction defined to IMS as non-response mode typically does not require an output message to the I/O PCB, therefore no output message is created and nothing is queued on the TPIPE.

- Response mode transaction

  The IMS application program associated with a transaction defined to IMS as non-response mode typically will insert an output message to the I/O PCB. Because the IMS resource adapter does not expect a response from a SYNC_SEND interaction, the output message, if inserted, is queued on the TPIPE with the name of the generated clientID. However, interactions SYNC_RECEIVE_ASYNCOUTPUT_SINGLE_NOWAIT or SYNC_RECEIVE_ASYNCOUTPUT_SINGLE_WAIT can be used to retrieve the response, if performed following the SYNC_SEND interaction and in the same application and on the same connection.

- Non-response mode or response mode transactions that invoke an IMS application program that inserts to an alternate PCB

  A message inserted to an alternate PCB can be retrieved by executing an interaction on a dedicated persistent socket connection. This can be done by the following ways:

  1. Ensuring that the connectionFactory used by the interaction is configured with a value of TRUE for the CM0Dedicated property.
  2. Providing the following values for the interaction:
     - IMSInteractionSpec property interactionVerb=SYNC_RECEIVE_ASYNCOUTPUT_SINGLE_NOWAIT or SYNC_RECEIVE_ASYNCOUTPUT_SINGLE_WAIT
     - IMSInteractionSpec property commitMode=0

– IMSConnectionSpec property clientID= the name of the alternate PCB

Dedicated persistent socket processing model

The following scenarios describe a SYNC_SEND interaction on a dedicated persistent socket connection for different types of transactions. SYNC_SEND interactions use commitMode0 and dedicated persistent socket connections can only be used for commitMode 0 interactions.

- Non-response mode transaction

  The IMS application program associated with a transaction defined to IMS as non-response mode typically does not insert an output message to the I/O PCB, therefore no output message is created and nothing is queued on a TPIPE.

- Response mode transaction

  The IMS application program associated with a transaction defined to IMS as non-response mode typically will insert an output message to the I/O PCB. Because the IMS resource adapter does not expect a response from a SYNC_SEND interaction, the output message, if inserted, is queued on the TPIPE with the name provided for the clientID of the interaction. Messages queued to this type of TPIPE can be retrieved by issuing a SYNC_RECEIVE_ASYNCOUTPUT_SINGLE_NOWAIT or SYNC_RECEIVE_ASYNCOUTPUT_SINGLE_WAIT interactions. The TPIPE name is the clientID specified for the SYNC_SEND interaction. clientID is required for interactions that use a dedicated persistent socket connection.

- Non-response mode or response mode transactions that invoke an IMS application that inserts to an alternate PCB

  A message inserted to an alternate PCB can be retrieved by executing an interaction on a dedicated persistent socket connection. This can be done by the following ways:

  1. Ensuring that the connectionFactory used by the interaction is configured with a value of TRUE for the CM0Dedicated property.
  2. Providing the following values for the interaction:
     – IMSInteractionSpec property interactionVerb=SYNC_RECEIVE_ASYNCOUTPUT_SINGLE_NOWAIT or SYNC_RECEIVE_ASYNCOUTPUT_SINGLE_WAIT
     – IMSInteractionSpec property commitMode=0
     – IMSConnectionSpec property clientID= the name of the alternate PCB

SYNC_SEND_RECEIVE programming model

To run a transaction in IMS, your Java application executes a SYNC_SEND_RECEIVE interaction. Your application provides a value of SYNC_SEND_RECEIVE for the interactionVerb property and a value of 0 or 1 for the commitMode property of the IMSInteractionSpec object used by the execute method. However, the SYNC_SEND_RECEIVE interaction processing is different for shareable and dedicated persistent socket connections.

Shareable persistent socket processing model

The following scenarios describe the SYNC_SEND_RECEIVE interaction on a shareable persistent socket during normal processing, error processing, and execution timeout. These steps apply for both commit mode 1 and commit mode 0.

- Normal processing scenario

  The IMS resource adapter, with the application server, obtains either an available connection from the connection pool or creates a new connection. The IMS resource adapter, as part of initalizing a new connection generates a clientID for the connection. The generated clientID identifies the socket connection, and in the case of commit mode 0 interactions, the TPIPE and associated OTMA Asynchronous Hold Queue.

The IMS resource adapter ensures that a socket is associated with the connection and sends the request with input data to IMS Connect using that socket. IMS Connect then sends the message to IMS, where IMS runs the transaction and returns the output message.

For commit mode 0 interactions, on receiving the output message, the IMS resource adapter internally sends an ACK message to IMS which signals IMS to discard the output from the IMS queue. When the client application closes the connection or terminates, the connection is returned to the connection pool for reuse by other commit mode 0 or commit mode 1 interactions.

- Error processing scenario

  All errors result in a resource exception being thrown to the client application. In addition, some errors result in the socket being disconnected by IMS Connect. In the case of commit mode 0 interactions, an exception means the output message cannot be delivered to the client application. However, following exceptions undelivered output messages for commit mode 0 interactions on shareable persistent socket connections can be retrieved if the SYNC_SEND_RECEIVE interaction specified that undelivered output should be rerouted to a specific destination. To have an undelivered output message rerouted to a specific destination, the following additional properties must be specified in the IMSInteractionSpec object passed on the SYNC_SEND_RECEIVE interaction:

  – The purgeAsyncOutput property must be set to FALSE so that undelivered output is not purged
  – The reRoute property must be set to TRUE and a reroute destination specified in the RouteName property

  To retrieve undelivered output from a reroute destination, a separate client application issues a SYNC_RECEIVE_ASYNCOUTPUT_SINGLE_NOWAIT or SYNC_RECEIVE_ASYNCOUTPUT_SINGLE_WAIT interaction on a dedicated persistent socket connection, providing the reroute destination as the clientID of the interaction.

  Note: The default value of the purgeAsyncOutput property is TRUE.

  When purgeAsyncOutput is TRUE, the following output messages are purged:

  – Undelivered output message inserted to the I/O PCB by the primary IMS application program.
  – Output messages inserted to the I/O PBC by secondary IMS application programs invoked by program to program switch.

  A value of FALSE for the PurgeAsyncOutput property should only be used if the reroute destination is specified.

- ExecutionTimeout scenario

  If an execution timeout occurs, the socket connection remains open but the output message is not delivered to the client application. However, following an execution timeout exception, undelivered output messages for commit mode 0 interactions on shareable persistent socket connections can be retrieved in either of the following two ways:

  – The same client application that issued the SYNC_SEND_RECEIVE interaction can issue a SYNC_RECEIVE_ASYNCOUTPUT_SINGLE_NOWAIT or SYNC_RECEIVE_ASYNCOUTPUT_SINGLE_WAIT interaction.
  – The undelivered output message can be rerouted to a specific destination as described in the error processing scenario above.

  When the client application closes the connection or terminates, the connection is returned to the connection pool so it can be reused by other commit mode 0 or commit mode 1 interactions.

Dedicated persistent socket processing model

Dedicated persistent socket connections can only be used for commit mode 0 interactions. The following scenarios describe the commit mode 0 SYNC_SEND_RECEIVE interaction on a dedicated persistent socket during normal processing, error processing, and execution timeout.

- Normal processing scenario

  Under normal circumstances, when a commit mode 0 SYNC_SEND_RECEIVE interaction is executed by a client application, the application server returns an existing connection with the user-specified

clientID, or creates a new connection with the user-specified clientID. The user-specified clientID identifies the socket connection and the TPIPE and associated OTMA Asynchronous Hold Queue.

The IMS resource adapter ensures that a socket is associated with the connection and sends the request with input data to IMS Connect using that socket. IMS Connect then sends the message to IMS, where IMS runs the transaction and returns the output message. On receiving the output message, the IMS resource adapter internally sends an ACK to IMS which signals to discard the output from the IMS queue. When the connection is closed or the application terminated, the connection is returned to the connection pool for reuse by another application that is running a commit mode 0 interaction with the same user-specified clientID.

- Error processing scenario

  All errors result in a resource exception being thrown to the client application. In addition, some errors result in the socket being disconnected by IMS Connect. In the case of commit mode 0 interactions, this means the output message cannot be delivered to the client application. The undelivered output is queued to the TPIPE associated with the user-specified clientID.

  The properties, purgeAsyncOutput and reRoute are not applicable to dedicated persistent sockets. You can not purge or reroute undelivered output messages on a dedicated persistent socket.

- ExecutionTimeout scenario

  If an execution timeout occurs, the socket remains open and the output of the commit mode 0 interaction is queued to the TPIPE associated with the user-specified clientID for later retrieval. When the connection is closed or the application terminated, the IMSManagedConnection object is returned to the connection pool for reuse by another application that is running a commit mode 0 interaction with the same user-specified clientID.

Retrieving asynchronous output

There are two types of socket connections, shareable persistent socket and dedicated persistent socket, that can be used to retrieve asynchronous output. The way to retrieve asynchronous output messages is different depending on the type of socket connection used. The interactionVerb property values that can be used to retrieve asynchronous output are: SYNC_RECEIVE_ASYNCOUTPUT_SINGLE_NOWAIT, and SYNC_RECEIVE_ASYNCOUTPUT_SINGLE_WAIT (along with the older SYNC_RECEIVE_ASYNCOUTPUT).

Note: There is no difference in function between SYNC_RECEIVE_ASYNCOUTPUT and SYNC_RECEIVE_ASYNCOUTPUT_SINGLE_NOWAIT. However, it is recommended that you use the new name SYNC_RECEIVE_ASYNCOUTPUT_SINGLE_NOWAIT with V9.1.0.1 and later deliverables of the IMS resource adapter. Only the new name, SYNC_RECEIVE_ASYNCOUTPUT_SINGLE_NOWAIT, will be used in the rest of this document.

The difference between SYNC_RECEIVE_ASYNCOUTPUT_SINGLE_NOWAIT and SYNC_RECEIVE_ASYNCOUTPUT_SINGLE_WAIT determines how IMS Connect checks for output on the IMS OTMA Asynchronous Hold Queue. For SYNC_RECEIVE_ASYNCOUTPUT or SYNC_RECEIVE_ASYNCOUTPUT_SINGLE_NOWAIT interactions, if there is no asynchronous output in the IMS OTMA Asynchronous Hold Queue when the retrieve request is made, IMS Connect will return an execution timeout notification as soon as the execution timeout value specified by the client application has passed. For this reason, the shortest possible execution timeout value, 10, is recommended for SYNC_RECEIVE_ASYNCOUTPUT_SINGLE_NOWAIT interactions.

For a SYNC_RECEIVE_ASYNCOUTPUT_SINGLE_WAIT interaction, if there is no asynchronous output in the IMS OTMA Asynchronous Hold Queue when the retrieve request is made, IMS Connect will wait up to the length of time specified in the executionTimeout property of the interaction for OTMA to return a message. If there is still no asynchronous output in the hold queue when the execution timeout has passed, IMS Connect will return an execution timeout error. For a SYNC_RECEIVE_ASYNCOUTPUT_SINGLE_WAIT interaction, you should select an appropriate execution timeout value, rather than the shortest possible value.

All three interactionVerb property values require commit mode 0 and can be used on both shareable persistent socket and dedicated persistent socket connections. In addition, IMSInteractionSpec properties purgeAsycOutput, reRoute and reRouteName do not apply to interactions which use these three interactionVerbs and are ignored by IMS Connector for Java. The way that interactionVerb properties are invoked on dedicated and shareable persistent socket connections is different.

Retrieving asynchronous output on dedicated persistent socket connections

To retrieve the queued output message on a dedicated persistent socket, the client application must execute a commit mode 0 interaction with the interactionVerb property of IMSInteractionSpec set to SYNC_RECEIVE_ASYNCOUTPUT_SINGLE_NOWAIT, or SYNC_RECEIVE_ASYNCOUTPUT_SINGLE_WAIT.

In addition to executing a commit mode 0 interaction on a dedicated persistent socket connection with the appropriate interactionVerb property of IMSInteractionSpec, the client application must also provide a value for the clientID property of IMSConnectionSpec. The clientID is required because it determines the TPIPE from which the asynchronous output will be retrieved. To retrieve output messages from a commit mode 0 interaction on a dedicated persistent socket, the clientID specified on the SYNC_RECEIVE_ASYNCOUTPUT_SINGLE_WAIT/NOWAIT interaction must match the value specified for the original commit mode 0 interaction. To retrieve output messages sent to an alternate PCB, the clientID specified on the SYNC_RECEIVE_ASYNCOUTPUT_SINGLE_WAIT/NOWAIT interaction must match the name of the alternate PCB. To retrieve output messages which were rerouted to a reRouteName destination, the clientID on the SYNC_RECEIVE_ASYNCOUTPUT_SINGLE_WAIT/NOWAIT interaction must be set to that reRouteName property destination.

Retrieving asynchronous output on shareable persistent socket connection

To retrieve an undelivered output message resulting from an interaction on a shareable persistent for which the reRoute flag has not been set to TRUE, the client application must execute a SYNC_RECEIVE_ASYNCOUTPUT_SINGLE_NOWAIT or SYNC_RECEIVE_ASYNCOUTPUT_SINGLE_WAIT interaction on the same shareable persistent socket connection in the same application that invoked the interaction that led to the asynchronous output being queued. The reason that the two interactions must be invoked within the same client application is that the IMS resource adapter automatically generates a client-ID for shareable persistent socket connections. This generated clientID identifies the socket connection as well as the associated OTMA TPIPE to which the asynchronous output is queued. A new client-ID is generated when a new connection is established. On shareable persistent socket connections, the clientID is generated by IMS Connector for Java and is unique for each connection. Therefore, to retrieve asynchronous output for a specific generated clientID, a connection with the same clientID must be used. This means that, for shareable persistent socket connections (which always have unique generated clientIDs) the same connection must be used. The only way to guarantee that the same connection will be used is to execute both interactions (the original interaction as well as the SYNC_RECEIVE_ASYNCOUTPUT_SINGLE_WAIT or SYNC_RECEIVE_ASYNCOUTPUT_SINGLE_NOWAIT interactions) within the same client application.

The following situations can result in an output message being queued on an IMS OTMA Asynchronous Hold Queue:

1. An IMS application program inserts an output message to an alternate PCB.
2. The output from a commit mode 0 interaction on a shareable or dedicated persistent socket cannot be delivered to the client application.
3. An interaction spawns a program-to-program switch for which the secondary output is not delivered to the client application. Secondary output is always commit mode 0 output.

Do not specify a value for the clientID property of IMSConnectionSpec for interactions on shareable persistent socket connections. On shareable persistent socket connections a user specified clientID is not allowed since IMS Connector for Java automatically generates the clientID.

Displaying output message counts

Using IMS Connect commands, you can choose to display output message counts. This topic describes how to display those message counts.

In IMS and OTMA terminology, a transaction pipe (TPIPE) is a logical connection between a client, such as IMS Connect, and the server, such as IMS OTMA. For commit mode 0 interactions, the TPIPE name is the clientID used for the interaction. For commit mode 0 interactions the IMS OTMA Asynchronous Hold Queue associated with the TPIPE has the same name as the clientID.

For commit mode 1 interactions, the TPIPE name is the IMS Connect port number used for the interaction, or in the case of Local Option the TPIPE name is the word, LOCAL. Therefore, each port will have a TPIPE which will be used for all clients running commit mode 1 interactions on that port.

You can use the IMS Connect command `/DISPLAY TMEMBER` *`IMSConnect_Name`* `TPIPE ALL` to view counts of the output messages sent to IMS Connector for Java, as well as messages inserted to ALTPCBS (Alternate Program Communication Blocks). The following sample output is from a `/DISPLAY TMEMBER HWS1 TPIPE ALL` command. A brief description of the types of TPIPEs and counts for the command output is also provided.

```
DFS000I     MEMBER/TPIPE          ENQCT      DEQCT     QCT STATUS     IMS1
DFS000I     HWS1            IMS1
DFS000I    -9999        0   0  0     IMS1
DFS000I    -HWSMIJRC        2   2  0     IMS1
DFS000I    -CLIENT01        3   2  1     IMS1
DFS000I    -ALTPCB1        2   1  1     IMS1
DFS000I    -HWS$DEF        1          0              1     IMS1
DFS000I    -RRNAME        1   0         1     IMS1
```

Commit Mode 1 interactions on a shareable persistent socket

- The TPIPE name is the port number used for the interaction. For example, 9999.
- The enqueue count (ENQCT) and dequeue count (DEQCT) will be equal and the queue count (QCT) will be 0, because undelivered output messages are not recoverable for commit mode 1 transactions.

Commit Mode 0 interactions on a shareable persistent socket

- The TPIPE name is generated by IMS Connector for Java and will have a prefix of ″HWS″. For example, HWSMIJRC.
- The enqueue count (ENQCT) and dequeue count (DEQCT) will be equal and the queue count (QCT) will be 0 if all messages are delivered to IMS Connector for Java.
- If output messages are not delivered to IMS Connector for Java on SYNC_SEND_RECEIVE interactions and the default values of reRoute FALSE and purgeAsyncOutput TRUE are used, the enqueue count (ENQCT) and dequeue count (DEQCT) will be equal and the queue count (QCT) will be 0. All undelivered output messages are discarded.
- If output messages are not delivered to IMS Connector for Java on SYNC_SEND_RECEIVE interactions and reRoute is set to TRUE and purgeAsyncOutput is set to FALSE, then the enqueue count (ENQCT) will be greater than the dequeue count (DEQCT) and the queue count (QCT) will be the number of messages that were not delivered to IMS Connector for Java. The TPIPE name is the value specified for the reRouteName property; for example, RRNAME, or a default value; for example, HWS$DEF.
- For SYNC_SEND interactions, output is not expected, so undelivered output does not apply. If SYNC_RECEIVE_ASYNCOUTPUT, SYNC_RECEIVE_ASYNCOUTPUT_SINGLE_NOWAIT and SYNC_RECEIVE_ASYNCOUTPUT_SINGLE_WAIT interactions are unsuccessful, the queue count does not change.

Commit Mode 0 interactions on a dedicated persistent socket

- Typically, the TPIPE name is provided by the Java application and will not have a prefix of ″HWS″. For example, CLIENT01. However, you may occasionally see a TPIPE name of ″HWS$DEF″. This is the default value for the reRouteName property.
- The enqueue count (ENQCT) and dequeue count (DEQCT) will be equal and the queue count (QCT) will be 0 if all messages are delivered to IMS Connector for Java, and no undelivered messages were rerouted from interactions on shareable persistent socket connections.
- If output messages are not delivered to IMS Connector for Java or rerouted from interactions on shareable persistent socket connections, the enqueue count (ENQCT) will be greater than the dequeue count (DEQCT) and the queue count (QCT) will be the number of messages that were not delivered. The TPIPE name is the user specified clientID name, for example, CLIENT01.

Output messages inserted to ALTPCBs (Alternate Program Communication Blocks)

- The TPIPE name is the name of the Alternate PCB. For example, ALTPCB1.

Chapter 8. Transaction processing

The topics in this section describe how the IMS resource adapter supports global transaction management and two-phase commit processing so that your application can run in a J2EE-compliant application server to access IMS transactions. The topics included are:

Global transaction support with two-phase commit

To protect and maintain the integrity of your critical business resources, IMS Connector for Java, as a J2EE Connector Architecture resource adapter, supports global transaction management and two-phase-commit processing. Using this support, you can build a J2EE application to group a set of changes into one transaction, or a single unit of work, so that all changes within a transaction are either fully completed or fully rolled back. This enables your application to run in a J2EE-compliant application server (for example, WebSphere Application Server) to access IMS transactions and data in a coordinated manner. Global transaction management ensures the integrity of the data in IMS.

Example of global transaction support

When you make changes to your protected resources, you want to guarantee that the changes are made correctly. For example, as a bank customer you want to transfer money from your savings account to your checking account. You want to be sure that when the money is deducted from your savings account it is added to your checking account simultaneously. You would not want this transaction to be completed only partially with the money deducted from your savings account but not added to your checking account.

In another example, you need to buy a ticket from San Francisco to Paris but a direct flight is not available. Unless you can successfully reserve a ticket from San Francisco to Chicago and another ticket from Chicago to Paris, you will not commit to your trip to Paris. That is, you will ″roll back″ your decision to go to Paris because having a confirmed seat for only one part of your trip is not useful to you.

In both of these examples, several smaller transactions are required in order to complete one overall transaction. If there is a problem with one of these smaller transactions, you would not want to commit the overall transaction (such as transferring money or going to Paris). Instead, you would want to roll back every step of the transaction so that none of the smaller transactions are committed. To transfer your money or to go on your trip to Paris successfully, you want the smaller transactions to be managed and coordinated together to complete the overall transaction.

To ensure a coordinated transaction process, the J2EE platform (which consists of a J2EE application server, J2EE application components, and a J2EE connector architecture resource adapter) provides a distributed transaction processing environment where transactions are managed transparently and resources are updated and recovered across multiple platforms in a coordinated manner.

Global transaction and two-phase commit support process

A J2EE-compliant application server (such as WebSphere Application Server) uses a Java transaction manager, also known as an *external coordinator*, to communicate with the application components (for example, Java servlets or Enterprise Java Beans) and the resource managers (for example, IMS or DB2®) through the resource adapters (for example, IMS Connector for Java) to coordinate a transaction.

If a transaction manager coordinates a transaction, that transaction is considered a global transaction. If a transaction manager coordinates a transaction with more than one resource manager, the external coordinator uses two-phase commit protocol.

In the previous bank example, you want to transfer money from your savings account to your checking account. If your savings account information resides on a separate resource manager from your checking account information (for example, your saving account resides on IMS and your checking account resides on DB2), the transaction manager in the application server (WebSphere Application Server) helps the application to coordinate the changes between IMS and DB2 transparently using two-phase commit processing. Specifically, the transaction manager works with the IMS resource adapter to coordinate the changes in IMS.

IMS Connector for Java is designed to work together with the Java transaction manager in the J2EE platform, the Resource Recovery Services (RRS) of z/OS, and IMS Connect to make consistent changes to IMS and other protected resources.

To participate in two-phase commit processing with IMS, IMS Connector for Java uses the IMS OTMA Synchronization level sync-point protocol. To participate in global transaction and two-phase commit processing when the changes are requested from a remote application, IMS uses RRS on z/OS.

RRS, from the point of view of IMS, acts as the ″external coordinator″ or sync-point manager to coordinate the update and recovery of resources. IMS Connector for Java and IMS Connect, interact with the Java transaction manager running in the application server and RRS on z/OS to allow a global transaction running on a J2EE platform to participate in a coordinated update with IMS running on the host.

When setting up a J2EE application to participate in a global transaction, you must select one of the two available communication protocols to be used between IMS Connector for Java and IMS Connect. The two communication protocols supported by IMS Connector for Java and IMS Connect are TCP/IP and Local Option.

Global transaction with TCP/IP

In a global transaction scope, your J2EE application component can access an IMS transaction by establishing a TCP/IP connection with IMS Connect. Underlying, IMS Connector for Java interacts with the Java transaction manager using the X/Open (XA) protocol to manage the global transaction and two phase commit processing. The XA protocol defines a set of interfaces and interactions describing how the Java transaction manager and the resource managers interact in a distributed transaction processing environment. IMS Connector for Java, together with IMS Connect, uses the XA protocol and works with IMS and Resource Recovery Services (RRS) on z/OS to make consistent changes.

Restrictions: You are required to have RRS running on the same MVS system with IMS Connect.

To set up RRS on IMS Connect, refer to *IMS Connect Guide and Reference* (SC27-0946). For more information about TCP/IP communication protocol for global transaction and two-phase commit processing, see Platform considerations and communication protocol considerations and Two-phase commit environment considerations.

Global transaction with Local Option

If your J2EE application component is running on WebSphere Application Server for z/OS, you can submit IMS transaction messages using Local Option and participate in global transaction processing. This transaction processing is coordinated by Resource Recovery Services (RRS) on z/OS and WebSphere Application Server for z/OS. IMS Connector for Java is RRS-compliant and is designed specifically to work with RRS so that the Java transaction manager in WebSphere and IMS, as the resource manager, can work together to make consistent changes to multiple protected resources. The XA protocol is not used by IMS Connector for Java when running global transaction with Local Option.

Restriction:

- To run a global transaction with Local Option, WebSphere Application Server for z/OS, IMS Connect, and IMS must run in the same MVS system.

Recommendation:

- Use Local Option for optimal performance.
- If WebSphere Application Server for z/OS is running on a different MVS than IMS and IMS Connect, you must use TCP/IP for global transaction.

For information about Local Option communication protocol for global transaction and two-phase commit processing, see Platform considerations and communication protocol considerations, Two-phase commit prerequisites, and Two-phase commit environment considerations.

Additional information on transaction support

Local Transaction

The J2EE Connection Architecture defines the javax.resource.cci.LocalTransaction interface to allow a resource manager, rather than a transaction manager, to coordinate a transaction locally. However, IMS Connector for Java only supports transaction coordination with a transaction manager. Thus, IMS Connector for Java does not support the javax.resource.cci.LocalTransaction interface. If you call the IMSConnection.getLocalTransaction() method you will get a NotSupportedException. To use transaction support with IMS Connector for Java, you need to either use the JTA transaction interface, or set an appropriate transaction attribute in the deployment descriptor in your application. See Using global transaction support in your application for more information.

One-phase commit processing

IMS Connector for Java supports one-phase commit optimization with the transaction manager. As a result, if all changes inside a transaction scope belong to the same IMS resource, the transaction manager might perform one-phase-commit optimization such that the transaction manager sends the phase two commit request directly to the resource manager for committing the changes without sending the phase one prepare request.

Non-global transaction processing

If no global transaction processing is used in the application (for example, when the transaction attribute is set to TX_NOTSUPPORTED), all non-global transaction processing uses ″Sync-On-Return″ (OTMA SyncLevel=None). By the time the IMS transaction is committed, the output has been returned to the client.

Conversational transaction processing in global transaction scope

IMS uses a conversational program to divide processing into a connected series of client-to-program-to-client interactions (also called iterations). Each iteration is a type of IMS conversational transaction. Conversational processing is used when one transaction contains several parts. Each part that comprises one large transaction is separately committed or rolled back.

You can run a conversational transaction in the global transaction scope if:

- Each iteration is run under the same transaction level. For example, if the first iteration is processed with a global transaction scope, then all the subsequent iterations in that IMS conversational transaction must be processed at a global transaction level. If you issue the second iteration with no transaction scope, IMS OTMA reports an error.
- Each iteration must be completed with a commit or rollback call before issuing the next iteration in the IMS conversation. You cannot group multiple iterations in a single global transaction scope.

For more information about using global transaction support, see the IMS Connector for Java web page at www.ibm.com/ims and go to Hints and Tips on the Support page.

Two-phase commit prerequisites

The prerequisites for two-phase commit processing with TCP/IP are:

- IMS Connector for Java 2.1.0 or later
- IMS Connect 2.1 or later
- RRS on z/OS 1.2 or later
- IMS Version 8 or later
- WebSphere Application Server Version for z/OS or distributed platforms 5.0 or later

The prerequisites for two-phase commit processing with Local Option are:

- IMS Connector for Java 1.2.2 or later
- IMS Connect 1.2 or later
- The RRS level associated with z/OS Version 2 Release 10 or later
- IMS Version 7 or later
- WebSphere Application Server 4.0.1, PTF 4 or later

Note: RRS must be installed and running for two-phase commit processing to occur. IMS and IMS Connect must also be enabled for RRS processing. (If you are using two-phase commit processing with Local Option, IMS Connect does not need to be enabled for RRS processing.) You can enable RRS processing on IMS Connect by either issuing the IMS Connect command, `SETRRS ON` or set `RRS=Y` in the IMS Connect configuration file. To ensure IMS is enabled with RRS, check that the RRS value in the startup parameter within your IMS environment is set to `Y`. This will appear in the job logs generated when IMS is brought up.

Additionally, to run two-phase commit IMS, IMS Connect, and RRS must all be in the same MVS image. For more information about two-phase commit, see Two-phase commit environment considerations.

Using global transaction support in your application

The J2EE platform allows you to use either a programmatic or a declarative transaction demarcation approach to manage transactions in your application. The programmatic approach is the component-managed transaction and the declarative transaction demarcation approach is the container-managed transaction.

Component-managed (or Bean-managed) transaction

The J2EE application uses the JTA javax.transaction.UserTransaction interface to demarcate a transaction boundary to a set of changes to the protected resource programmatically. Component-managed transactions can be used in both the servlet and the EJB environment. In the case of an EJB, you set the transaction attribute in its deployment descriptor as TX_BEAN_MANAGED.

A transaction normally begins with a UserTransaction.begin() call. When the application component is ready to commit the changes, it invokes a UserTransaction.commit() call to coordinate and commit the changes. If the application component must roll back the transaction, it invokes UserTransaction.rollback() and all changes are backed out. For example:

```
// Get User Transaction
javax.transaction.UserTransaction transaction =
ejbcontext.getUserTransaction();

// Start transaction
```

```
transaction.begin();

// Make changes to the protected resources.
// For example, use the J2EE/CA's CCI Interaction interface
// to submit changes to an EIS system(s)
interaction.execute(interactionSpec, input, output);

if (/* decide to commit */) {
// commit the transaction
transaction.commit();

} else { /* decide to roll back */
// rollback the transaction
transaction.rollback();
}
```

Container-managed transaction

Container-managed transactions can be used only in the EJB environment. The EJB specifies a container-managed transaction declaratively through the transaction attribute in the deployment descriptor (such as TX_REQUIRED). A container-managed transaction is managed by the EJB container. The container calls the appropriate methods (such as begin, commit, or rollback) on behalf of the EJB component. This declarative approach simplifies the programming calls in the EJB.

Related Reading: For more information about the J2EE architecture and JTA specifications, see http://java.sun.com/j2ee/docs.html.

Two-phase commit environment considerations

To run a two-phase commit application, consider the following suggestions:

- It is best to have as many MPP regions as possible running to ensure that two-phase commit applications do not contend for a region; because a transaction that is within a two-phase commit application uses an MPP region for the duration of the entire two-phase commit transaction.
- If a number of IMS transactions are performed within a two-phase commit transaction, at least that many MPP regions must be available to avoid hanging the two-phase commit application.
- To safeguard against a transaction that may be waiting for an extensive amount of time for resources, it is recommended to set an appropriate timeout value for each interaction taking place within the global transaction.
- Avoid having an excessive number of database interactions performed in one two-phase commit transaction. If multiple IMS transactions are used within a two-phase commit transaction, they could possibly contend or lock in an attempt to update or modify the same data. To avoid this, it's best to write an application that will prevent a user from accessing duplicate entries within the same two-phase commit operation.
- Consider configuring your IRLM or PI locking manager to use a block size that is as small as the smallest entry to that database. Larger block sizes might have two transactions contending for entries that may not even be the same and yet reside close to one another on the hard disk.
- If multiple interactions are performed using the same IMS transaction on the same IMS database within a global transaction (unit of work), each interaction with that IMS transaction must run on a separate MPP region. The IMS transaction must have a SCHDTYP=PARALLEL and a PARLIM=0 value, to indicate that the IMS transaction can run on multiple MPP regions and that it will always meet the scheduling requirements (the number of messages will be greater than zero) to process every interaction on a new MPP region.
- If a region is hung, waiting for RRS-OTMA and no execution timeout value has been set, you can end the attempt to run a transaction that is hanging the MPP region. This can be done by issuing a stop region IMS command with the abend transaction parameter. For example, /STOP REGION *reg*#ABDUMP *tranname*. This will rollback the transaction for that particular interaction and free the MPP region.

For more information about two-phase commit, including sample applications, see the *IMS Connector for Java Guide and Reference*.

Chapter 9. Diagnosing problems

The topics in this section provide information on how to log and trace component information and diagnose problems, as well as list the messages and exceptions of the IMS resource adapter and MFS plugin. The topics included are:

Diagnosing problems when using the IMS resource adapter

If you are unable to access IMS from your Java application, consider performing the following actions to diagnose the problem:

- Verify that you have the correct prerequisites for using the IMS resource adapter. See Prerequisites for using the IMS resource adapter.
- Verify that IMS Connect is active by ensuring that the outstanding IMS Connect reply ″HWSC0000I *IMS CONNECT READY* *ims_connect_name*″ appears on the system console of the target machine.
- Verify that the PORT and DATASTORE are ACTIVE by entering the IMS Connect command VIEWHWS at the IMS Connect outstanding reply.
- Verify that IMS is active by ensuring that the outstanding IMS reply ″DFS996I *IMS READY*″ appears on the system console of the target machine.
- Verify that the XCF status of both the IMS and IMS Connect members is ACTIVE by entering the IMS command /DISPLAY OTMA at the outstanding IMS reply. The display output should be similar to the following:

```
DFS000I     GROUP/MEMBER      XCF-STATUS   USER-STATUS
SECURITY    IMS1

DFS000I     XCFGRPNM
    IMS1

DFS000I     -IMSNAME          ACTIVE       SERVER           FULL
    IMS1

DFS000I     -ICONNAME         ACTIVE       ACCEPT TRAFFIC
    IMS1

DFS000I     *02033/143629*   IMS1
```

- If you’re using TCP/IP to communicate between the Java application and IMS Connect, verify that you can successfully ″ping″ the target host machine. If you cannot ping the host machine and you are using a host name rather than an IP address, ensure that the host name is sufficiently qualified.

If your IMS service is not providing the expected output from the IMS transaction, ensure that the output message returned by the IMS application program matches the output COBOL definition used by the service. For a J2EE application, you can view the IMS OTMA message containing the message returned by the IMS application program by setting the traceLevel property to 3. See Logging and tracing with the IMS resource adapter for instructions on how to turn on the IMS resource adapter trace. For more information on the IMS OTMA message, go to the IMS web site, http://www.ibm.com/ims and select **IMS Connect**.

Logging and tracing with the IMS resource adapter

The IMS resource adapter, in addition to other J2EE components, provides controls for logging and tracing component information. When these controls are set for logging and tracing and you run your Java application using the WebSphere Unit Test Environment, a trace file is created.

Note: Ensure that only one client is running when the trace is on.

To set controls for logging and tracing, complete the following steps:

1. In the Server Configuration view, double-click your **server configuration** to open the **WebSphere Server Configuration** editor.
2. Select the **J2C** tab in the editor.
3. On the **J2C Options** page, select an IMS resource adapter in the **J2C Resource Adapters** table.
4. Scroll down to the **J2C Connection Factories** table and select the connection factory for which you want to turn the trace on.
5. Scroll down to the **Resource Properties** table and select the TraceLevel resource property. Specify a non-zero value to enable logging and tracing. **TraceLevel** values correspond to constants in the interface com.ibm.connector2.ims.ico.IMSTraceLevelProperties.

TraceLevel Value	IMSTraceLevelProperties	Description
0	RAS_TRACE_OFF	No tracing or logging occurs.
1	RAS_TRACE_ERROR_EXCEPTION	Only errors and exceptions are logged.
2	RAS_TRACE_ENTRY_EXIT	Errors and exceptions plus the entry and exit of important methods are logged.
3	RAS_TRACE_INTERNAL	Errors and exceptions, the entry and exit of important methods, and the contents of buffers sent to and received from IMS Connect are logged.

6. After entering the TraceLevel value on the page for the **J2C** tab, select the **Trace** tab.
7. Ensure that the **Enable trace** check box is selected. To enable logging and tracing in the IMS Resource adapter, enter the following in the **Trace string** field:

   ```
   com.ibm.connector2.ims.*=all=enabled
   com.ibm.ims.ico.*=all=enabled
   ```

   Other combinations of trace strings will enable tracing in other components. For example, with the following trace string:

   ```
   com.ibm.ejs.j2c.*=all=enabled:com.ibm.connector2.*=all=enabled
   ```

   the string `com.ibm.ejs.j2c.*` provides you with logging and tracing of WebSphere's implementation of the J2EE Connector Architecture and the string `com.ibm.connector2.*` provides you with logging and tracing of all of the resource adapters, including IMS.
8. You can accept the default name and location of the trace output file or you can modify it. For example, depending on how you set your substitution variables, the default name and location might be:

   ```
   your_workspace\.metadata\.plugins\com.ibm.etools.server.core
   \tmp0\logs\server1\trace.log
   ```

   To modify the default name and location, enter a different name and location of the file in the **Trace output file** field on the **Trace Options** page of the server configuration.
9. When you are finished making changes, close the editor and select **Yes** to save your changes.
10. Check the **Status** column of the **Servers** view and restart the server instance, if necessary. You will most likely have to restart the server instance if you are using the WebSphere Test Environment.

11. Run your Java application and then examine the trace file.

J2CA0056I, WLTC0017E, HWSP1445E, and HWSSL00E Error Messages

J2CA0056I

When IMS resource adapter throws an exception, it can be caught by a component other than your Java application. For example, when you run a deployed application, IMS Connector for Java exceptions are often caught by the WebSphere Application Server. WebSphere Application Server may then issue its own message, including in it the message from the IMS resource adapter exception. For example, when execution timeout occurs, you see the following on the Console:

- ```
  J2CA0056I: The Connection Manager received a fatal connection error
  from the
  Resource Adaptor for resource myConnFactry.  The exception which
  was received is
  ICO0080E:
  com.ibm.connector2.ims.ico.IMSTCPIPManagedConnection@e59583c.
  processOutputOTMAMsg(byte[],IMSInteractionSpec, int) error.
  Execution timeout has occurred for this interaction.
  The executionTimeout was [0] milliseconds.  The IMS Connect TIMEOUT
  was used.
  ```

J2CA0056I is an informational message from WebSphere Application Server. The fatal connection error refers to the fact that IMS Connect closes the socket in the case of an execution timeout, which results in WebSphere Application Server's Connection Manager removing the connection object for the socket from the connection pool.

Another example occurs when a transaction (non-persistent) socket is used for a commit mode 0 interaction. In this case, you see the following on the Console:

- ```
  J2CA0056I: The Connection Manager received a fatal connection error
  from the
  Resource Adaptor for resource  myConnFactry.  The exception which
  was received is
  ICO0089I:
  com.ibm.connector2.ims.ico.IMSTCPIPManagedConnection@6db5d83a.call(Connection,
  InteractionSpec, Record, Record). Non-persistent socket closed for
  Commit Mode 0 IMS transaction.
  ```

J2CA0056I is an informational message from WebSphere Application Server. The fatal connection error refers to the fact that IMS Connect closes the transaction socket and the IMS resource adapter causes WebSphere Application Server's Connection Manager to remove the connection object for the socket from the connection pool.

WLTC0017E

A local transaction containment (LTC) is used to define the application server behavior in an unspecified transaction context. For example, if a single method within a container managed EJB that has a transaction attribute of NotSupported is called outside of any transaction scope, WebSphere will create a local transaction to handle resources used during the execution of that method. The message above is produced by the WebSphere Transaction Monitor to indicate that the resources enlisted with the LTC were rolled back instead of committed due to setRollbackOnly() being called on the LTC. This message does not require any action by the user and is for your information only.

- ```
  WLTC0017E: Resources rolled back due to setRollbackOnly() being
  called.
  ```

Note: The prefix of a WebSphere Application Server message indicates the component that issued the message. You can find documentation of these messages, by component, in Integration Edition's Help using **WebSphere Application Server Enterprise > Quick reference > Messages**. All messages are documented with user/system action and explanation. These messages are also documented in the
  ```

WebSphere Application Server Version 5 Information Center.

HWSP1445E

When you provide Connection Properties to the New IMS Service wizard in Integration Edition or when you configure a Connection Factory for use by your Java application, you choose whether or not you are using SSL with the **SSLEnabled** property. If you are using SSL (**SSLEnabled=TRUE**), then the port number you provide must be configured as an SSL port in IMS Connect. If you accidentally provide a non-SSL port for your Java application, unexpected results will occur when you run your application.

- IMS Connector for Java will throw an exception indicating a communication error:

```
javax.resource.spi.CommException:
ICO0003E:
com.ibm.connector2.ims.ico.IMSTCPIPManagedConnection@56503fc6.connect()
error.
Failed to connect to host [CSDMEC13], port [9999].
[java.net.SocketException:
Connection reset by peer: socket closed]
```

- The following IMS Connect message will be displayed on the MVS console:

```
HWSP1445E UNKNOWN EXIT NAME SPECIFIED IN MESSAGE PREFIX; MSGID=
   /9  * !hR, M=SDRC
```

The first step in establishing an SSL connection involves the SSL handshake protocol, in which the client (IMS Connector for Java) sends the server (IMS Connect) an SSL ″Hello″ message. In the scenario described above, IMS Connect is waiting for an incoming message on a non-SSL port. When IMS Connect receives the handshake message it interprets it as an OTMA message with a valid Exit name in the prefix and issues message HWSP1445E.

HWSSSL00E

The opposite scenario to the one above occurs when you are **not** using SSL (**SSLEnabled=FALSE**), but the port number you provide for your Java application is configured as an SSL port in IMS Connect. In this case:

- IMS Connector for Java will throw an exception indicating a communication error:

```
javax.resource.spi.CommException: ICO0005E:
com.ibm.connector2.ims.ico.IMSTCPIPManagedConnection@5bcdcdd4.receive()
error. A communication error occurred while sending or receiving
the IMS message.
[java.net.SocketException: Connection reset by peer: socket closed]
```

- The following IMS Connect message will be displayed on the MVS console:

```
HWSSSL00E Unable to initialize the SSL socket:Error while reading
or writing data
```

IMS Connect's attempt to initialize the SSL socket fail, since it does not receive the initial client ″Hello″ message that is part of the SSL handshake protocol.

IMS resource adapter messages and exceptions

While you develop Java programs that use IMS Connector for Java, you might encounter situations in which your program throws exceptions. Some of these exceptions are thrown by IMS Connector for Java, while others are thrown by class libraries used by IMS Connector for Java (such as the Java class libraries). This topic provides information on exceptions generated by IMS Connector for Java J2C applications.

The following terms, in *italics* in the message descriptions that follow, are replaced by specific values at runtime.

hostname
The TCP/IP host name of the machine that is running IMS Connect.

innermethodname
The name of the method that originally throws this exception. This exception has been caught by IMS Connector for Java and is being re-thrown to another exception, according to the Common Connector Framework specification.

length The length of the data.

libraryFileName
The Local Option native library file name.

llvalue
The value of LL.

maxlength
The maximum valid length of the data.

methodname
The name of the method that is throwing this exception.

mode The type of interaction between IMS Connector for Java and the IMS Connect component on the host (as defined in the interactionspec).

nativeMethodName
The Local Option native method name.

portnumber
The port number that is assigned to IMS Connect.

propertyname
The name of the property.

propertyvalue
The value of the property.

reasoncode
The reason code that is returned by IMS Connect.

rectype
The type of the record.

returncode
The return code, formatted in decimal, that is returned by IMS Connect.

sensecode
The sense code, formatted in decimal, that is returned from IMS OTMA

socketexception
The socket exception.

source_exception
The exception thrown when the error first occurred in an internal method.

source_methodname
The internal method in which the error first occurred.

state The internal state of IMS Connector for Java.

Related Reading

- For information on exceptions that are thrown from other class libraries, see the Javadoc information for the specific class library.
- For information on exceptions related to Local Option support, see *IBM IMS Connect User's Guide and Reference*. Some exceptions are thrown based on IMS, IMS OTMA or IMS Connect errors returned by

IMS Connect. For information on IMS OTMA and IMS Connect errors, see *IBM IMS Open Transaction Manager Access Guide* and *IBM IMS Connect User's Guide and Reference*, respectively. For information on IMS errors, see *IBM IMS Messages and Codes.*

Exceptions generated by IMS Connector for Java J2C applications

The following exception messages are produced by applications built with the Java 2 Platform, Enterprise Edition (J2EE) Connector Architecture (J2C) class libraries when an error condition is detected.

ICO0001E

```
javax.resource.spi.EISSystemException:
ICO0001E: methodname error.
IMS Connect returned error:
RETCODE=[returncode], REASONCODE=[reasoncode].
reasoncode_string.
```

Explanation: IMS Connect returned an error. The connection in error will not be reused. *reasoncode_string* provides a brief description of the *reasoncode* , if available.

User Action: Check the MVS console for associated IMS Connect error messages. IMS Connect error messages begin with the characters " **HWS**". For diagnostic information on the return code (*returncode*) and reason code (*reasoncode*) values, as well as IMS Connect error messages, see the *IMS Connect Guide and Reference.*

ICO0002E

```
javax.resource.spi.EISSystemException:
ICO0002E:methodname error.
IMS OTMA returned error:
SENSECODE=[sensecode], REASONCODE=[otmareasoncode].
[source_methodname:source_exception]
```

Explanation: IMS OTMA returned a NAK error.

User Action: For diagnostic information on the sense code (*sensecode*) and OTMA reason code (*otmareasoncode*) values of the NAK error, see the *IMS OTMA Guide and Reference.* Note that IMS Connector for Java displays *sensecode* and *otmareasoncode* in decimal. If the application is running with two-phase commit, you may receive the following sense code values with the NAK error:

- Sense code = 17 (decimal, 23 Hex)

  Your IMS is not enabled with RRS processing. Ensure your IMS has Protected Conversation processing with RRS enabled. See Two-phase commit prerequisites for more information.
- Sense code = 46 (decimal, 2E Hex)

  RRS and two-phase commit processing is not supported by IMS Connect and IMS Connector for Java. Make sure that both your IMS Connect and IMS Connector for Java is at least version 2.1.0 or above.

ICO0003E

```
javax.resource.spi.CommException:
ICO0003E:methodname error.
Failed to connect to host [hostname],
port [portnumber].
[java_exception]
```

Explanation: IMS Connector for Java was unable to connect to the host and port combination. *java_exception* indicates the reason for the failure to connect. For additional information see the User Action section below.

User Action: Examine *java_exception* to determine the reason for the failure to connect to the host. Some values for *java_exception* are:

- **java.net.UnknownHostException**: *hostname*

  The host name you specified when configuring the Connection Factory used by your application is invalid or your application specified an invalid host name. Check the spelling of the host name. You may have to use the fully qualified path for host name or the IP address.

- **java.net.ConnectException**: **Connection refused**

  Some possible reasons for the exception are:

  – The port number is invalid. Ensure that you are using a valid port number for the IMS Connect indicated by *hostname*.

  – The specified port is stopped. This can be determined using the IMS Connect command VIEWHWS. If the port is stopped its status will be NOT ACTIVE. Use the IMS Connect command, **OPENPORT** *dddd*, where *dddd* is the specified port number, to start the port.

  – IMS Connect on the specified host is not running. Start IMS Connect on the host machine.

  – TCP/IP was restarted without canceling and restarting IMS Connect or issuing **STOPPORT** followed by **OPENPORT** on the host.

- **java.net.SocketException**: **connect (code=10051)**

  Some possible reasons for the exception are:

  – The machine with the specified host name is unreachable on the TCP/IP network. Ensure that the host machine is accessible from the TCP/IP network. Verify by issuing the ping command to the specified host machine. Enter the ping command on the machine on which IMS Connector for Java is running. Start TCPIP on the host, if it is not started.

  – TCP/IP was restarted but the status of the port used by the application was NOT ACTIVE. To correct this situation you can do one of the following:

  –

    - Use the IMS Connect command OPENPORT *dddd*, where *dddd* is the port number, to activate the port
    - Restart IMS Connect

ICO0005E

```
javax.resource.spi.CommException:
ICO0005E:methodname error.
A communication error occurred while sending or receiving the IMS message.
[java_exception]
```

Explanation: IMS Connector for Java was unable to successfully complete a send and receive interaction with the target IMS Connect. *java_exception* indicates the reason for the failure to complete the interaction. For additional information see the User Action section below.

User Action: Examine *java_exception* to determine the reason for the failure. Some values for *java_exception* are:

- **java.io.EOFException**

  Some possible reasons for the exception are:

  – The timeout value specified in the IMS Connect configuration file is exceeded before IMS Connect receives a response from IMS. Exceeding a timeout value typically occurs when there is no region available in IMS to run the IMS transaction that processes the client's request. If this is the case, ensure that an appropriate region is started and available to process the request. Exceeding a timeout value can also occur if the IMS application program associated with the transaction is stopped. If this is the case, use the IMS command /START PROGRAM to start the IMS application program.

  – **Note:** This is the expected behavior for the following configurations:

- Releases of IMS Connector for Java prior to 1.2.6, running with IMS Connect 1.2
- IMS Connector for Java 1.2.6 or 2.1.0, running with IMS Connect 1.2 plus APAR PQ71355

– A Java client tries to use a previously active client (for example, a connection from the pool) for which an IMS Connect **STOPCLNT** command has been issued.

- **java.net.SocketException: Connection reset by peer: socket write error**

  Some possible reasons for the exception are:

  – A Java client attempts to use a connection for which the underlying socket is no longer connected to IMS Connect. This can happen if IMS Connect is recycled, but the application server is not. After IMS Connect is restarted, the connection pool will contain connections that formerly were successfully connected to IMS Connect. As clients attempt to reuse each of these connections, the exception **java.net.SocketException** is thrown and the connection object removed from the connection pool. Eventually all these connections will be removed from the pool and new connections will successfully be created.

  – **Note:** This behavior can be changed in WebSphere Application Server by setting the **Purge Policy** of the connection factory used by the Java application to *Entire Pool*.

  – TCP/IP on the host is coming down.

ICO0006E

```
javax.resource.ResourceException:
ICO0006E:methodname error.
The value provided for DataStoreName is null or an empty string.
```

Explanation: The method indicated in *methodname* was invoked using an empty DatastoreName parameter. This error message will appear in the trace log when a connection factory with an empty DatastoreName parameter is started. This message will be followed by a J2EE Connector warning,

```
J2CA0007W: An exception occurred while invoking method setDataStoreName on
com.ibm.connector2.ims.ico.IMSManagedConnectionFactory used by resource
Connection_Factory_JNDI_name.
```

Processing will then continue leading to other error messages after IMS Connect sends a response indicating that a datastore with a null name cannot be found. The underlying message which triggers the other messages is:

```
javax.resource.spi.EISSystemException: ICO0001E:
com.ibm.connector2.ims.ico.IMSTCPIPManagedConnection@.processOutputOTMAMsg(byte[],
InteractionSpec, Record) error. IMS Connect returned error: RETCODE=[4],
REASONCODE=[NFNDDST ]. Datastore not found.
```

When this error occurs, a corresponding HWSS0742W warning message is displayed on the MVS console of the host machine where IMS Connect is running. This HWSS0742W message will include a field showing the datastore name that it attempted to find, in this case all blanks:

```
DESTID=        ;
```

User Action: Provide a valid name for the DatastoreName parameter. In a managed environment, the DatastoreName is specified when you are configuring a Connection Factory to be used by WebSphere Application Server. In a non-managed environment, the DatastoreName is specified in your Java application.

ICO0007E

```
javax.resource.NotSupportedException:
ICO0007E:methodname error.
The [propertyName] property value [propertyValue] is not supported.
```

Explanation: The value *propertyValue* specified for the property *propertyName* is not supported.

User Action: Provide a supported value for the named property. For example, certain values of the InteractionVerb property of the InteractionSpec class that are defined in the J2C architecture are not supported by the IMSInteractionSpec class in this release of IMS Connector for Java. Also the ReRoute value TRUE is not supported on dedicated persistent socket connections.

ICO0008E

```
javax.resource.ResourceException:
ICO0008E:methodname error. The value [propertyValue] of the [propertyName]
property exceeds the maximum allowable length
of [maxPropertyLength].
```

Explanation: The length of the value *propertyValue* supplied for property *propertyName* exceeds *maxPropertyLength*, the maximum length allowed for values of property *propertyName*.

User Action: Provide a value for the named property which does not exceed *maxPropertyLength*.

ICO0009E

```
javax.resource.ResourceException:
ICO0009E:methodname error.
The [propertyName] property value [propertyValue] is invalid.
```

Explanation: The value *propertyValue* specified for the property *propertyName* is not valid.

User Action: Provide a value which is valid for the named property. For example, valid values for the InteractionVerb property of the InteractionSpec class of IMS Connector for Java are listed in the Javadoc for the IMSInteractionSpec class.

ICO0010E

```
javax.resource.spi.IllegalStateException:
ICO0010E:methodname error.
Method invoked on invalid IMSConnection instance.
```

Explanation: The method indicated in *methodname* was invoked on an invalid IMSConnection instance. If the *methodname* is `lazyEnlist`, an attempt was made to enlist a connection in the current transaction that could not be enlisted.

User Action: The named method was most likely issued on an IMSConnection instance that was already closed.

- If the *methodname* is not `lazyEnlist`, ensure that the IMSConnection instance is not already closed before you attempt to use it or close it.
- If the *methodname* is `lazyEnlist`, ensure that your application is not using non-managed connections in a managed environment, as only managed connections are eligible for Lazy Transaction Enlistment Optimization. For more information please refer to the Deferred Enlistment topic found in the WebSphere Application Server 6.0 online information center.

ICO0011E

```
javax.resource.spi.IllegalStateException:
ICO0011E:methodname error.
Method invoked on invalid IMSInteraction instance.
```

Explanation: The method indicated in *methodname* was invoked on an invalid IMSInteraction instance.

User Action: The named method was most likely issued on an IMSInteraction instance that was already closed. Ensure that the IMSInteraction instance is not already closed before you attempt to use it or close it.

ICO0012E

```
javax.resource.ResourceException:
ICO0012E:methodname error.
The value provided for HostName is null or an empty string.
```

Explanation: The method indicated in *methodname* was invoked using a null or empty HostName parameter.

User Action: Provide a valid HostName parameter. In a managed environment, the property value is specified when you are configuring a Connection Factory to be used by WebSphere Application Server. In a non-managed environment, the property value is specified in your Java application.

ICO0013E

```
javax.resource.ResourceException:
ICO0013E:methodname error.
ConnectionManager is null.
```

Explanation: The method indicated in *methodname* was invoked. The application server invoked the **createConnectionFactory** method of the **IMSManagedConnectionFactory** class with a null ConnectionManager object.

User Action: Provide a valid HostName parameter. This form of the **createConnectionFactory** method is used in a managed environment. It is not typically invoked by a client program. Contact the service personnel for your application server.

ICO0014E

```
javax.resource.ResourceException:
ICO0014E:methodname error.
Input record contains no data.
```

Explanation: The method indicated in *methodname* was invoked with an input record that contained no data.

User Action: Verify that the input record that you provide is not empty.

ICO0015E

```
ResourceAdapterInternalException
ICO0015E:methodname error.
Unexpected error encountered while processing the OTMA message.
[java_exception]
```

Explanation: An unexpected internal error was encountered while processing the OTMA message.

User Action: Contact your IBM service representative.

ICO0016E

```
javax.resource.ResourceException:
ICO0016E:methodname error.
The value provided for DataStoreName is null or an empty string.
```

Explanation: The method indicated in *methodname* was invoked using an empty DatastoreName parameter. This error message will appear in the trace log when a connection factory with an empty DatastoreName parameter is started. This message will be followed by a J2EE Connector warning,

```
J2CA0007W: An exception occurred while invoking method setDataStoreName on
com.ibm.connector2.ims.ico.IMSManagedConnectionFactory used by resource
Connection_Factory_JNDI_name.
```

Processing will then continue leading to other error messages after IMS Connect sends a response indicating that a datastore with a null name cannot be found. The underlying message which triggers the other messages is:

```
javax.resource.spi.EISSystemException: ICO0001E:
com.ibm.connector2.ims.ico.IMSTCPIPManagedConnection@.processOutputOTMAMsg(byte [],
 InteractionSpec, Record) error. IMS Connect returned error: RETCODE=[4],
REASONCODE=[NFNDDST ].  Datastore not found.
```

When this error occurs, a corresponding HWSS0742W warning message is displayed on the MVS console of the host machine where IMS Connect is running. This HWSS0742W message will include a field showing the datastore name that it attempted to find, in this case all blanks:

```
DESTID=        ;
```

User Action: Provide a valid name for the DatastoreName parameter. In a managed environment, the DatastoreName is specified when you are configuring a Connection Factory to be used by WebSphere Application Server. In a non-managed environment, the DatastoreName is specified in your Java application.

ICO0017E

```
ResourceAdapterInternalException
ICO0017E:methodname error.
Invalid value provided for TraceLevel.
```

Explanation: An invalid trace level was specified.

User Action: Specify a valid trace level. Optionally, this exception can be ignored due to the fact that the default trace level will be used for this connection factory. In this case, the connection factory is still usable but the trace level will be the default trace level.

ICO0018E

```
javax.resource.ResourceException:
ICO0018E:methodname error.
The value provided for PortNumber is null.
```

Explanation: The method indicated in *methodname* was invoked using a null PortNumber.

User Action: Provide a valid PortNumber parameter. In a managed environment, the property value is specified when you are configuring a Connection Factory to be used by WebSphere Application Server. In a non-managed environment, the property value is specified in your Java application.

ICO0024E

```
javax.resource.ResourceException:
ICO0024E:methodname error.
Invalid segment length (LL) of [llvalue] in input object.[java_exception]
```

Explanation: The input message provided by the Java program for the IMS application program contains a value for its segment length which is either negative, 0, or greater than the number of bytes of data in the message segment.

User Action: Provide the correct value for the segment length of the input message.

ICO0025E

```
javax.resource.IllegalArgumentException:
ICO0025E:methodname error.
Invalid segment length (LL) of [llvalue] in OTMA message.
```

Explanation: The output message provided by the IMS application program contains a value for its segment length which is either negative, 0, or greater than the number of bytes of data in the message segment. The output message provided by the IMS application program is contained in the OTMA message.

User Action: Ensure that your IMS application program provides valid lengths for the segments of its output message.

ICO0026E

```
javax.resource.ResourceException:
ICO0026E:methodname error.
An error was encountered while processing the IMS message.
[source_methodname:source_exception]
```

Explanation: An error occurred while processing the IMS transaction input or output message. *source_exception* provides additional information regarding the cause of the error.

User Action: Examine *source_exception* for additional information regarding the cause of the error. Some suggested actions to take, based on the value of *source_exception* are:

- **java.io.IOException**

  Error preparing input or output record. Ensure that the objects you are providing to IMS Connector for Java for use as the IMS transaction input and output are defined properly for the J2C architecture. For example, ensure that they implement the interfaces **javax.resource.cci.Record** and **javax.resource.cci.Streamable**.
- **com.ibm.ims.ico.IMSConnResourceException**

  The OTMA message containing the IMS transaction output message contained an invalid length field (i.e., LLLL was <= 0). If this error continues to occur after verifying that your IMS application program is returning a valid output message, contact your IBM service representative.
- **java.lang.IllegalArgumentException**

  The output message returned from IMS Connect is invalid. Ensure that the release levels of IMS Connector for Java and IMS Connect are compatible. For example, if you built a transactional required EJB application to perform a two phase commit transaction via TCP/IP by using IMS Connector for Java version 2.1, but at runtime, you are using IMS Connect version 1.2 instead of version 2.1, you will receive this error message. Hereby, either you update to IMS Connect version 2.1 or create a none global transactional EJB application.

ICO0030E

```
javax.resource.spi.ApplicationServerInternalException:
ICO0030E:methodname error.
[source_methodname:source_exception]
```

Explanation:A runtime error or exception was detected in *methodname* during the interaction. *source_methodname:source_exception* indicates where the error or exception that was detected in *methodname* originally occurred and may provide additional information regarding the cause of the error.

User Action: Examine *source_exception* for additional information regarding the cause of the error. The action(s) to be taken depend on the value of *source_methodname:source_exception*. Some suggested actions to take, based on the value of *source_methodname:source_exception* are:

- java.lang.OutOfMemoryError

  This error is thrown when the Java Virtual Machine cannot allocate an object because it is out of memory, and no more memory could be made available by the garbage collector. Increase the amount of memory available to the virtual machine used by WAS.
- java.io.InterruptedIOException

An InterruptedIOException is thrown to indicate that an input or output transfer has been terminated because the thread performing it was terminated. Investigate reasons why the thread may have been terminated.

ICO0031E

```
javax.resource.spi.IllegalStateException:
ICO0031E:methodname error.
Protocol violation. The Interaction Verb [interactionverb] is not allowed for
the current state [state].
[java_exception]
```

Explanation: The interaction attempted by the application resulted in a protocol violation. **[***interactionverb***]** is the value of the interactionVerb property of the IMSInteractionSpec object that was used for the interaction. **[***state***]** is the current state of the protocol used for the interactions between IMS Connector for Java and IMS Connect.

For example, a protocol violation would occur if your Java program is not in conversation with IMS, but attempted an interaction with IMS using the SYNC_END_CONVERSATION value for the interactionVerb property.

User Action: Ensure that you are using an appropriate value for the interactionVerb property of IMSInteractionSpec. Check the IMS Connector for Java documentation for values of the interactionVerb property that are supported by IMS Connector for Java. A particular release of IMS Connector for Java may not support all the values defined by the J2EE Connector Architecture.

ICO0034E

```
javax.resource.NotSupportedException:
ICO0034E:methodname error.
Auto-commit not supported.
```

Explanation: Auto-commit is currently not supported by IMS Connector for Java.

User Action: Ensure that your Java application uses classes and methods that are appropriate for the level of support currently provided by IMS Connector for Java.

ICO0035E

```
javax.resource.NotSupportedException:
ICO0035E:methodname error.
Local Transaction not supported.
```

Explanation: Local Transactions are not currently supported by IMS Connector for Java.

User Action: Ensure that your Java application uses classes and methods that are appropriate for the level of support currently provided by IMS Connector for Java.

ICO0037E

```
javax.resource.NotSupportedException:
ICO0037E:methodname error.
ResultSet not supported.
```

Explanation: ResultSets are currently not supported by IMS Connector for Java.

User Action: Ensure that your Java application uses classes and methods that are appropriate for the level of support currently provided by IMS Connector for Java.

ICO0039E

```
javax.resource.spi.IllegalStateException:
ICO0039E:methodname error.
Not in CONNECT state.
```

Explanation: The sequence of interactions between IMS Connector for Java and IMS Connect is invalid. The current state of the protocol used for the interactions between IMS Connector for Java and IMS Connect is not CONNECT as it needs to be at this point in the interactions.

User Action:This is most likely an error in IMS Connector for Java or IMS Connect. Contact your IBM service representative.

ICO0040E

```
javax.resource.NotSupportedException:
ICO0040E:methodname error.
IMSConnector does not support this version of execute method.
```

Explanation: IMS Connector for Java does not support the form of the execute method that takes two input parameters and returns an object of type javax.resource.cci.Record.

User Action: Use the supported form of the execute method in class IMSInteraction. The supported form of the execute method has the following signature:

```
boolean execute(InteractionSpec, Record input, Record output)
```

ICO0041E

```
javax.resource.ResourceException:
ICO0041E:methodname error.
Invalid interactionSpec specified [interactionSpec].
```

Explanation: An invalid InteractionSpec object was passed to the execute method of class **com.ibm.connector2.ims.ico.IMSInteraction**.

User Action: Ensure that the InteractionSpec object that you pass to the execute method of class **com.ibm.connector2.ims.ico.IMSInteraction** is of type **com.ibm.connector2.ims.ico.IMSInteractionSpec**.

ICO0042E

```
javax.resource.ResourceException:
ICO0042E:  methodname error.
Input is not of type Streamable.
```

Explanation: The input object provided to the execute method of **com.ibm.connector2.ims.ico.IMSInteraction** for the ″input″ parameter was either null or did not implement the interface **javax.resource.cci.Streamable**. This exception most likely occurs when an application is written to use the J2EE Connector Architecture Common Client Interface (CCI). This exception should not occur if WebSphere Studio Application Developer Integration Edition is used to build the input message.

The execute method allows null input objects for some types of interactions. For example, interactions with interactionVerb values of SYNC_END_CONVERSATION and SYNC_RECEIVE_ASYNCOUTPUT allow null input objects.

User Action: Ensure that you are providing a valid javax.resource.cci.Record object for the ″input″ parameter to the execute method. For example, ensure that this object implements the interfaces **javax.resource.cci.Record** and **javax.resource.cci.Streamable**.

ICO0043E

```
javax.resource.ResourceException:
ICO0043E:  methodname error.
Output is not of type Streamable.
```

Explanation: The output object provided to the execute method of **com.ibm.connector2.ims.ico.IMSInteraction** was either null or did not implement the interface **javax.resource.cci.Streamable**. This exception most likely occurs when an application is written to use the J2EE Connector Architecture Common Client Interface (CCI). This exception should not occur if WebSphere Studio Application Developer Integration Edition is used to build the output message.

User Action: Ensure that you are providing a valid output object to the execute method.

ICO0044E

```
javax.resource.NotSupportedException:
ICO0044E:methodname error.
RecordFactory is not supported by IMS Connector for Java.
```

Explanation: RecordFactory is currently not supported by IMS Connector for Java.

User Action: Ensure that your Java application uses classes and methods that are appropriate for the level of support currently provided by IMS Connector for Java.

ICO0045E

```
javax.resource.NotSupportedException:
ICO0045E:  methodname error.
Invalid type of ConnectionRequestInfo.
```

Explanation: An invalid ConnectionRequestInfo object was passed to an IMS Connector for Java method.

User Action: This is most likely an error in IMS Connector for Java. Contact your IBM service representative.

ICO0049E

```
javax.resource.NotSupportedException:
ICO0049E:methodname error.
The security credentials passed to getConnection do not match existing
security credentials.
```

Explanation: The security credentials in the request do not match the security credentials of the IMSManagedConnection instance that was being used to process the request.

User Action: Contact your IBM service representative.

ICO0053E

```
javax.resource.ResourceException:
ICO0053E: methodname error.
Invalid clientID value. Prefix HWS is reserved by IMS Connector for Java.
```

Explanation: The value specified for the property clientID is invalid. The prefix 'HWS' is reserved by IMS Connector for Java.

User Action: Provide a valid value for clientID property. A valid value should follow the following rules:

- is not a null string;
- does not start with a blank field;
- does not start with IMS Connector for Java reserved prefix 'HWS';
- is 8 characters long;

- uses valid characters A - Z, 0 - 9, and @, #, $.

ICO0054E

```
javax.resource.ResourceException:
ICO0054E:methodname error.
Invalid ConnectionSpec.
```

Explanation: IMS Connector for Java was unable to cast the connectionSpec provided for this connection to type IMSConnectionSpec. While the Common Client Interface will accept a connectionSpec object for any supported connector, IMS Connector for Java will only work with an IMSConnectionSpec or a derivative of IMSConnectionSpec as its connectionSpec.

User Action: Ensure that the connectionSpec used by your application is an IMSConnectionSpec or inherits from IMSConnectionSpec.

ICO0055E

```
javax.resource.ResourceException:
ICO0055E:methodname error.
Failed to cast the connection object to IMSConnection.
```

Explanation: IMS Connector for Java was unable to cast the connection object allocated by the ConnectionManager for this connection to type IMSConnection. IMS Connector for Java will only work with an IMSConnection or a derivative of IMSConnection as its connection object. This error might be the result of a problem with the ConnectionManager.

User Action: Please contact your IBM service representative.

ICO0056E

```
javax.resource.ResourceException:
ICO0056E:methodname error.
IMSConnectName is only valid for Local Option connections
which can only be used in z/OS or OS/390.
```

Explanation: Setting the IMSConnectName property of an IMSManagedConnectionFactory instance is mandatory for managed connection factory to be used for Local Option connections. Furthermore, you can only use Local Option to communicate with IMS Connect if your application using IMS Connector for Java is running on the z/OS or OS/390 platform. This exception indicates that you have a specified a value for the IMSConnectName property but your application is not running on neither the z/OS nor OS/390 platforms.

User Action: Ensure that your application using IMS Connector for Java is running on the z/OS or OS/390. Note that it is also required that your application (or more precisely, the Web server where your application is running) must be running in the same MVS image as IMS Connect. If this is not the case, for example, if you plan to run your application on a workstation platform or if the Web server where you plan to run your application is on z/OS but in a different MVS image than IMS Connect, ensure that the connection factory used by your application is set up to use TCP/IP communication.

ICO0057E

```
javax.resource.spi.IllegalStateException:
ICO0057E:methodname error.
Invoked with invalid connection handle.
```

Explanation: The application is in an illegal state: the connection handle (IMSConnection instance) used for this interaction is not valid. This could occur if the application attempted to use a connection handle for a previously used connection or the handle for the wrong connection if the application has more than one connection open.

User Action: Ensure that the application is using the currently valid IMSConnection instance for that connection.

ICO0058E

```
javax.resource.ResourceException:
ICO0058E:methodname error.
Interactions SYNC_SEND_RECEIVE, SYNC_SEND, SYNC_RECEIVE_ASYNCOUTPUT,
SYNC_RECEIVE_ASYNCOUTPUT_SINGLE_NOWAIT and
SYNC_RECEIVE_ASYNCOUTPUT_SINGLE_WAIT interactions with Commit Mode 0
are not supported with Local Option.
```

Explanation: You can use Local Option to communicate with IMS Connect only if your application using IMS Connector for Java with the selection of Commit Mode 1.

User Action: Ensure that your application using IMS Connector for Java is selected with Commit Mode 1. If you plan to run your application with Commit Mode 0, correct your application to use TCP/IP communication.

ICO0059E

```
javax.resource.ResourceException:
ICO0059E: methodname error.
SYNC_END_CONVERSATION interation with Commit Mode 0 is not supported.
```

Explanation: Interaction SYNC_END_CONVERSATION with Commit Mode0 is not supported.

User Action: IMS Connector for Java supports the interaction combination SYNC_END_CONVERSATION with Commit Mode 1, SYNC_SEND_RECEIVE with Commit Mode 0, and SYNC_RECEIVE_ASYNCOUTPUT with Commit Mode 0.

ICO0060E

```
java.lang.UnsatisfiedLinkError:
ICO0060E:methodname error.
Error loading Local Option native library: libname=libraryFileName.
[source_exception].
```

Explanation: The Local Option native library cannot be found in any of the directories listed in the libpath.

User Action: Ensure that the Local Option native library exists in one of the directories in the LIBPATH environment variable. If you are running IMS Connector for Java in WebSphere Application Server for z/OS and OS/390, ensure that the full name of the directory that contains the Local Option native library file is defined in the LIBPATH environment variable for your J2EE server. For more information, see "Preparing the base operating system" in the WebSphere Application Server Version 6.0 Information Center .

ICO0061E

```
javax.resource.ResourceException:
ICO0061E:methodname error.
Local Option runs only in z/OS and OS/390.
```

Explanation: You can use Local Option to communicate with IMS Connect only if your application using IMS Connector for Java is running on the z/OS or OS/390 platform.

User Action: Ensure that your application using IMS Connector for Java is running on the z/OS or OS/390. Note that it is also required that your application (or more precisely, the Web server where your application is running) must be running in the same MVS image as IMS Connect. If this is not the case, for example, if you plan to run your application on a workstation platform or if the Web server where

you plan to run your application is on z/OS but in a different MVS image than IMS Connect, ensure that the connection factory used by your application is set up to use TCP/IP communication.

ICO0062E

```
javax.resource.ResourceException:
ICO0062E:methodname error.
Error loading Local Option native method: libfilename=libraryFileName,
methodname=nativeMethodName. [source_exception].
```

Explanation: The Local Option native method cannot be found.

User Action: Verify that you have the correct level of IMS Connector for Java resource adapter and Local Option native library installed on your system. Always use the version of the Local Option native library that shipped with the IMS resource adapter that you installed in your WebSphere Application Server for z/OS and OS/390 system. See ″Prerequisites for using IMS Connector for Java″ for more information.

ICO0063E

```
javax.resource.spi.ResourceAdapterInternalException:
ICO0063E:methodname error.
Exception thrown in native method. [source_exception].
```

Explanation: An internal error occurred in the Local Option native method.

User Action: Contact your IBM service representative.

ICO0064E

```
javax.resource.spi.SecurityException:
ICO0064E:methodname error.
Invalid security credential.
```

Explanation: The subject provided by WebSphere Application Server did not contain a security credential available that is supported by IMS Connector for Java.

User Action: Ensure that you have the correct level of WebSphere Application Server for z/OS and OS/390 installed. See the ″ Prerequisites for using IMS Connector for Java″ section for details. Configure WebSphere Application Server for z/OS and OS/390 to provide a security credential that is supported by IMS Connector for Java. IMS Connector for Java supports the PasswordCredential for TCP/IP connections and the UToken GenericCredential for Local Option connections.

ICO0065E

```
javax.resource.spi.SecurityException:
ICO0065E:methodname error.
Error obtaining credential data from the security credential.[source_exception].
```

Explanation: There was a security related error in obtaining the credential data from the security credential provided by the application server.

User Action: Ensure that you have correctly set up security for your application server so that the user associated with the calling program is authorized to extract the data from a security credential.

ICO0066E

```
javax.resource.ResourceException:
ICO0066E:methodname error. Error loading WebSphere Application Server
Transaction Manager. [source_exception].
```

Explanation: An error occurred when accessing the Transaction Manager of the WebSphere Application Server for processing the transaction request.

User Action: Ensure that you have the correct level of WebSphere Application Server for z/OS and OS/390 installed. See the ″ Prerequisites for using IMS Connector for Java″ section for details.

ICO0068E

```
javax.resource.ResourceException:
ICO0068E:methodname error.
Error obtaining the transaction object.  [java_exception]
```

Explanation: An error occurred while attempting to determine if a transaction has been started using the WebSphere Application Server Transaction Manager.

User Action: Ensure that you have the correct level of WebSphere Application Server for z/OS and OS/390 installed. See the ″Prerequisites for using IMS Connector for Java″ section for details.

ICO0069E

```
javax.resource.spi.ResourceAllocationException
ICO0069E:methodname error.
Error obtaining RRS transaction context token.
IMSConnResourceException: RRS retcode=[rrs_routinecode].
```

Explanation: An unexpected internal error occurred while obtaining an RRS transaction context token for processing the global transaction.

User Action: Check the RRS job log for associated RRS error messages. For diagnostic information on the RRS return code (*rrs_routinecode*) see the *MVS Programming: Resource Recovery* manual for your release of z/OS or OS/390.

ICO0070E

```
javax.resource.spi.EISSystemException
ICO0070E:methodname error.
IMS Connect reported an RRS error: IMS Connect Return Code=[returncode],
RRS Routine name=[rrs_routine], RRS Return code=[rrs_routinecode]."
```

Explanation: IMS Connect returned an error resulting from an RRS failure.

User Action: Check the MVS console for associated IMS Connect and RRS error messages. For diagnostic information on the return code (*returncode*) value, as well as IMS Connect error messages, see the *IMS Connect Guide and Reference*. For diagnostic information on the RRS return code (*rrs_routinecode*) locate the RRS routine name (*rrs_routine*) within the *MVS Programming: Resource Recovery* manual for your release of z/OS or OS/390.

ICO0071E

```
javax.transaction.xa.xAException
ICO0071E:methodname error.
A communication error occurred when processing the XA
commandtype operation. [java_exception]
```

Explanation: There are numerous reasons why a communication failure could have occurred during the processing of a global transaction. A TCP/IP or socket failure could have taken place or IMS Connect could have been brought down. The connection in error will not be reused.

User Action: Examine the *java_exception* to determine the reason for the failure to connect to the host. Also check the MVS console for associated IMS Connect or TCP/IP error messages. Validate that machine can be reached through TCP/IP and that IMS Connect has not been brought down. The command type (*commandtype_string*) displayed in the error message refers to the stage at which this communication failure occurred during the global transaction: prepare, commit, rollback, recover, or forget.

ICO0072E

```
javax.transaction.xa.xAException:
ICO0072E:methodname error.
The associated UR for the Xid is not found.
```

Explanation: During transaction processing a UR that was tied to a specific Xid was eliminated by manual intervention or an error in IMS Connect or RRS.

User Action: Refer to the *WebSphere Application Server InfoCenter Reference Library* for steps on how to acquire transaction information and Xids within the WebSphere Application Server logs. Refer to the *IMS Connect Guide and Reference* for IMS Connect commands that will list out the Xid and their associated UR. Verify that a UR is listed for that Xid. Verify that the global transaction was not left in a heuristic state.

ICO0073E

```
javax.transaction.xa.xAException:
ICO0073E:methodname error.
RRS is not available.
```

Explanation: RRS has been brought down or communication between RRS and IMS Connect has ended.

User Action: Check the MVS console for associated IMS Connect and RRS error messages. Ensure that RRS has not been brought down on your z/OS or OS/390 system. Refer to the *IMS Connect Guide and Reference* for IMS Connect commands that can be used to verify that it is RRS enabled.

ICO0074E

```
javax.transaction.xa.xAException:
ICO0074E: The RRS rrs_routine call returns with a return code [rrs_routinecode].
```

Explanation: During the processing of your global transaction the following RRS error message was passed in by IMS Connect.

User Action: Check the MVS console for associated IMS Connect and RRS error messages. For diagnostic information on the RRS return code (*rrs_routinecode*) locate the RRS routine name (*rrs_routine*) within the *MVS Programming: Resource Recovery* manual for your release of z/OS or OS/390.

ICO0075E

```
javax.transaction.xa.xAException:
ICO0075E:methodname error.
The transaction branch may have been heuristically completed. [rrs_exception]
```

Explanation: An RRS error has been passed in by IMS Connect that indicates that the processing of your transaction may have been affected in such a way as to leave it in a heuristic situation. It reveals a possibility that part of the transaction committed and part of it encountered an error during the commit phase which may have prevented it from committing. The *rrs_exception* is an ICO0074E error message indicating the RRS routine and return code associated with this issue.

User Action: Refer to the documentation of the ICO0074E error for more information regarding the RRS error message. Refer to the *WebSphere Application Server InfoCenter Reference Library* for steps on how to acquire transaction information and Xids within the WebSphere Application Server logs. Refer to the *IMS Connect Guide and Reference* for IMS Connect commands that will list out the Xid and their associated UR. Determine the Xid and URs involved and the result that should have been committed to IMS. Verify values within IMS to ensure that a heuristic state has occurred. A decision must then be made to take actions to rectify the data within IMS so that it matches the result that would have been committed or to rectify the other databases involved to return to a state prior to the execution of that transaction.

ICO0076E

```
javax.resource.ResourceException:
ICO0076E:methodname error. An internal error occurred. [rrs_exception]
```

Explanation: An internal error occurred while trying to extract information about an RRS error message passed in by IMS Connect. The *rrs_exception* is an ICO0074E error message indicating the RRS routine and return code associated with the error.

User Action: Refer to the documentation of the ICO0074E error for more information regarding the RRS failure that has taken place. Please contact your IBM service representative.

ICO0077E

```
javax.resource.ResourceException:
ICO0077E:methodname error. The transaction has already rolled back. [rrs_exception]
```

Explanation: An RRS error has been passed in by IMS Connect that indicates the attempt to rollback a transaction has been made a second time upon the same UR. RRS will prevent the second rollback from taking place and throw an error indicating that such an action is being attempted. The *rrs_exception* is an ICO0074E error message indicating the RRS routine and return code associated with the error.

User Action: No action is needed as the transaction should be rolled back. Refer to the documentation of the ICO0074E error for more information regarding the RRS failure that has taken place. As a precaution, verify that data prior to the execution of the transaction has not been lost or modified.

ICO0078E

```
javax.resource.ResourceException:
ICO0078E: methodname error.
A valid user-specified clientID is required for interactions on a dedicated persistent connection.
```

Explanation: A valid, user-specified value is required for the clientID property when a value of 0 is specified for the commitMode property, and the interaction is using a dedicated persistent socket connection. This applies to SYNC_SEND_RECEIVE, SYNC_SEND, SYNC_RECEIVE_ASYNCOUTPUT, SYNC_RECEIVE_ASYNCOUTPUT_SINGLE_NOWAIT and SYNC_RECEIVE_ASYNCOUTPUT_SINGLE_WAIT interactions.

User Action: Provide a valid value for the clientID property. A valid value should follow the following rules:

- is not a null string
- does not start with a blank field
- does not start with IMS Connector for Java reserved prefix 'HWS'
- is 8 characters long
- has valid characters A - Z, 0 - 9, and @, #, $

ICO0079E

```
com.ibm.connector2.ims.ico.IMSDFSMessageException:
ICO0079E:methodname error.
IMS returned DFS message:DFS_message
```

Explanation: IMS returned the message indicated by *DFS_message* instead of the output of the IMS transaction. This exception is thrown if the interaction uses the value IMS_REQUEST_TYPE_IMS_TRANSACTION for the imsRequestType property of IMSInteractionSpec.

For example, if the Java application attempts to run an IMS transaction that is stopped, this exception is thrown and the value of *DFS_message* is

```
DFS065 hh:mm:ss TRAN/LTERM STOPPED
```

User Action: Find the explanation and response that corresponds to *DFS_message* in the *IMS Messages and Codes* documentation, then address the problem in IMS.

ICO0080E

```
javax.resource.spi.EISSystemException:
ICO0080E:methodname error.
Execution timeout has occurred for this interaction. The executionTimeout
was [executionTimeout_value] milliseconds. The IMS Connect
TIMEOUT was used.
```

Explanation: The time it took for IMS Connect to send a message to IMS and receive the response was greater than the IMS Connect TIMEOUT value. The IMS Connect TIMEOUT value is:

- Specified in the IMS Connect configuration member for SYNC_SEND_RECEIVE interactions
- Two seconds for SYNC_RECEIVE_ASYNCOUTPUT, SYNC_RECEIVE_ASYNCOUTPUT_SINGLE_NOWAIT, and SYNC_RECEIVE_ASYNCOUTPUT_SINGLE_WAIT interactions

The reason of IMS Connect TIMEOUT value has been used is the executionTimeout property for this interaction was not specified or has been set to zero.

User Action: Ensure your application has set a valid executionTimeout value. To set the executionTimeout values, you can either use WebSphere Studio or use the setExecutionTimeout method. For detail instruction, please refer to the topic of Setting execution timeout values in WebSphere Studio Application Developer Integration Edition 5.0.1 Help.

ICO0081E

```
javax.resource.spi.EISSystemException:
ICO0081E:methodname error.
Execution timeout has occurred for this interaction. The executionTimeout
value specified was [executionTimeout_value] milliseconds.
The value used by IMS Connect was
[rounded_executionTimeout_value] milliseconds.
```

Explanation: The time it took for IMS Connect to send a message to IMS and receive the response was greater than the executionTimeout value that was rounded to an appropriate execution timeout interval. Once a valid execution timeout value is set, this value is converted into a value that IMS Connect can use.

User Action: If the rounded execution timeout value is not what you expected, please verify with the follow table of conversion rules:

Range of user-specified values	Conversion rule
1 - 250	If the user-specified value is not divisible by 10, it is converted to the next greater increment of 10.
251 - 1000	If the user-specified value is not divisible by 50, it is converted to the next greater increment of 50.
1001 - 60000	The user-specified value is converted to the nearest increment of 1000. Values that are exactly between increments of 1000 are converted to the next greater increment of 1000.
60001 - 3600000	The user-specified value is converted to the nearest increment of 60000. Values that are exactly between increments of 60000 are converted to the next greater increment of 60000.

For more examples, please refer to the topic of Valid execution timeout values in WebSphere Studio Application Developer Integration Edition 5.0.1 Help.

ICO0082E

```
javax.resource.NotSupportedException:
ICO0082E:methodname error.
Execution timeout has occurred for this interaction. The executionTimeout
value of [{executionTimeout_value}] milliseconds is not supported.
The valid range is [{executionTimeout_waitforever_flag}, 0 to
{maximum_executionTimeout_value}] milliseconds.
The IMS Connect TIMEOUT was used.
```

Explanation: The execution timeout value specified for the executionTimeout property was above or below the minimum or maximum timeout values respectively.

User Action: Ensure that your application has set a valid value for executionTimeout property. The execution timeout value is represented in milliseconds and must be a decimal integer in the range of 1 to 3600000, inclusively. Also it could be -1 if you want an interaction to run without a time limit.

ICO0083E

```
javax.resource.ResourceException::
ICO0083E:methodname error.
SYNC_SEND_RECEIVE, SYNC_SEND, SYNC_RECEIVE_ASYNCOUTPUT,
SYNC_RECEIVE_ASYNCOUTPUT_SINGLE_NOWAIT and
SYNC_RECEIVE_ASYNCOUTPUT_SINGLE_WAIT interactions with Commit Mode 0
are not valid within the scope of a global transaction.
```

Explanation: SYNC_SEND_RECEIVE, SYNC_SEND, SYNC_RECEIVE_ASYNCOUTPUT, SYNC_RECEIVE_ASYNCOUTPUT_SINGLE_NOWAIT and SYNC_RECEIVE_ASYNCOUTPUT_SINGLE_WAIT interactions with Commit Mode 0 are not valid within the scope of a global transaction. Because currently the global transaction requires SYNC_LEVEL_SYNCPOINT and SYNC_LEVEL_SYNCPOINT only valid with Commit Mode 1.

User Action:

- If you want to use Commit Mode 0, ensure that your application is configured as a ″non-transactional″ application.
- If you want to run your interactions within the scope of a global transaction, then the commitMode property value must be 1.

ICO0084E

```
javax.resource.ResourceException:
ICO0084E:methodname error.
An unexpected internal IMS Connector for Java error occurred.
[source_method] [source_exception]
```

Explanation: A PrivelegedActionException occurred while executing a *[source_method]* call in *methodname*. This exception will occur if Java 2 security is enabled and the user associated with the calling program, *methodname*, or any program in the current call stack is not authorized to execute *[source_method]*.

User Action: Ensure that you have correctly set up security for your application server so that the user associated with the calling program plus any programs in the current call stack at the time of the exception is/are authorized to execute *[source_method]*. Alternatively, you could turn off Java 2 security checking in the application server.

ICO0085E

```
javax.resource.ResourceException:
ICO0085E: methodname error.
Protocol violation. A user-specified clientID is not allowed for interactions
on a shareable persistent socket.
```

Explanation: The value specified for clientID property is not allowed. Because the connection factory is configured for shareable persistent socket, a user-specified clientID is not allowed within this kind of connection factory.

User Action: For shareable persistent socket connection factory, IMS Connector for Java provides generated clientID. User-specified clientID is not allowed. To determine if you are using a shareable persistent socket, check for a value of FALSE for the CM0Dedicated property of the connection factory used by the interaction.

ICO0086E

```
javax.resource.ResourceException::
ICO0086E:methodname error.
Invalid value was specified for CommitMode property.
```

Explanation: The CommitMode value you have specified in the commitMode property field is invalid.

User Action: Ensure that your application has set a valid value for commitMode property. Values supported are:

- Value 1 (SEND_THEN_COMMIT), indicates that IMS processes the transaction and sends a response back before committing the data.
- Value 0 (COMMIT_THEN_SEND), indicates that IMS processes the transaction and commits the data before sending a response.

ICO0087E

```
javax.resource.ResourceException:
ICO0087E: methodname error.
Protocol violation. Commit Mode 1 is not allowed for interactions on a
dedicated persistent socket.
```

Explanation: The value 1 specified for Commit Mode property is invalid. Because the connection factory is configured for dedicated persistent socket, Commit Mode 1 is not allowed within this kind of connection factory.

User Action: For dedicated persistent socket connection factory, Commit Mode 0 interactions are valid. To determine if you are using a dedicated persistent socket check for a value of TRUE for the CM0Dedicated property of the connection factory used by the interaction.

ICO0088E

```
javax.resource.ResourceException:
ICO0088E: methodname error.
Protocol violation. SYNC_RECEIVE_ASYNCOUTPUT interactions are not allowed
on a shareable persistent sockets.
```

Explanation: The value SYNC_RECEIVE_ASYNCOUTPUT specified for interactionVerb property is invalid. Because the connection factory is configured for shareable persistent socket, SYNC_RECEIVE_ASYNCOUTPUT is not allowed within this kind of connection factory.

User Action: SYNC_SEND_RECEIVE, SYNC_SEND, and SYNC_END_CONVERSATION are the valid values for the interactionVerb property for interactions on a shareable persistent connection. To determine if you are using a shareable persistent connection, check for a value of FALSE for the CM0Dedicated property of the connection factory used by the interaction.

ICO0089I

```
javax.resource.ResourceException::
ICO0089I: methodname.
 Non-persistent socket closed for Commit Mode 0 IMS transaction.
```

Explanation: Running CommitMode 0 with non-persistent socket (transaction socket), IMS Connector for Java will force removal of managed connection object from Connection Pool.

User Action: This is not an error message, no action required.

ICO0091E

```
javax.resource.ResourceException:
ICO0091E: methodname
error.SSL client context could not be created. [{1}]
```

Explanation: An SSL Context could not be created due to one of the following reasons:

- The algorithm used to check the integrity of the keystore cannot be found
- The certificates in the keystore could not be loaded
- The key cannot be recovered (e.g. the given password is wrong).

User Action: Ensure the following:

- The algorithm used to create certificates must be one that is supported by IBMJSSE.
- The passwords for the keystore and truststore are correct.

ICO0096I

```
javax.resource.ResourceException:
ICO0096I: methodname
Warning. Invalid value provided for SSL parameter.
```

Explanation: The method indicated in *methodname* was invoked using a null or empty SSLKeystoreName, SSLKeystorePassword, SSLTruststoreName or SSLTruststorePassword parameter. This is an informational message to let the user know that one of the above-mentioned parameters is a null or an empty string. This will not terminate the program execution.

User Action: Provide valid values for SSLKeystoreName, SSLKeystorePassword, SSLTruststoreName and SSLTruststorePassword parameters. For convenience, private keys and certificates can be stored either in a keystore or a truststore. Therefore only one set of valid values (either SSLKeystoreName and SSLKeystorePassword or SSLTruststoreName and SSLTruststorePassword) are required for proper execution.

ICO0097E

```
javax.resource.ResourceException:
ICO0097E:methodname error.
{0} error. The given value is invalid for 'SSLEncryptionType'.
The value must be 'STRONG' for strong encryption or 'WEAK'
for weak encryption.
```

Explanation: A value other than strong or weak was provided for the SSLEncryptionType parameter.

User Action: Provide either strong or weak for the SSLEncryptionType parameter. The value is not case-sensitive.

ICO0111E

```
javax.resource.ResourceException:
ICO0111E:methodname error.
 SSLEnabled must be set to FALSE when using Local Option.
```

Explanation: The property IMSConnectName is set to a non-null value and the property SSLEnabled is set to true. However, SSL is not supported on local option connections (which is indicated by providing a value for IMSConnectName parameter).

User Action: Set SSLEnabled to false.

ICO0113E

```
javax.resource.spi.CommException:
ICO0113E: methodname error.
Socket Timeout has occurred for this interaction. The Socket Timeout value
specified was [socket timeout value] milliseconds.
[source_exception:exception_reason]
```

Explanation: The time for IMS Connector for Java to receive a response from IMS Connect is greater than the time specified for Socket Timeout.

User Action: Ensure that the time value of Socket Timeout is sufficient for IMS Connector for Java to receive a response from IMS Connect. If it is not, increase the value. If the value of Socket Timeout given is sufficient, it is possible that network problems are causing delays. Contact your network administrator.

ICO0114E

```
javax.resource.ResourceException:
ICO0114E: methodname error.
The Socket Timeout Property value of [socket timeout value] is invalid.
[source_exception:exception_reason]
```

Explanation: The value [socket timeout value] specified for the Socket Timeout property is not valid.

User Action: Review the exception_reason provided. Ensure a positive numerical value was given for Socket Timeout.

ICO0115E

```
javax.resource.spi.CommException:
ICO0115E: methodname error.
A TCP Error occurred.
```

Explanation: This is an error in the underlying protocol.

User Action: Contact your network administrator.

ICO0117E

```
javax.resource.ResourceException:
ICO0117E: methodname error.
Protocol violation: Commit Mode 1 is not allowed for SYNC_SEND,
SYNC_RECEIVE_ASYNCOUTPUT, SYNC_RECEIVE_ASYNCOUTPUT_SINGLE_NOWAIT
and SYNC_RECEIVE_ASYNCOUTPUT_SINGLE_WAIT interactions.
```

Explanation: The IMS resource adapter currently only supports Commit Mode 0 for SYNC_SEND interactions.

User Action: Commit Mode 0 is required for SYNC_SEND, SYNC_RECEIVE_ASYNCOUTPUT, SYNC_RECEIVE_ASYNCOUTPUT_SINGLE_NOWAIT and SYNC_RECEIVE_ASYNCOUTPUT_SINGLE_WAIT , SYNC_RECEIVE_ASYNCOUTPUT, SYNC_RECEIVE_ASYNCOUTPUT_SINGLE_NOWAIT and SYNC_RECEIVE_ASYNCOUTPUT_SINGLE_WAIT interactions. Commit Mode 1 is valid with SYNC_SEND_RECEIVE and SYNC_END_CONVERSATION interactions.

ICO0118E

```
javax.resource.ResourceException:
ICO0118E: methodname error.
Protocol violation. IMS request type 2(IMS_REQUEST_TYPE_IMS_COMMAND)
is not allowed for SYNC_SEND, SYNC_END_CONVERSATION, SYNC_RECEIVE_ASYNCOUTPUT,
SYNC_RECEIVE_ASYNCOUTPUT_SINGLE_NOWAIT and SYNC_RECEIVE_ASYNCOUTPUT_SINGLE_WAIT
interactions.
```

Explanation: The value 2(IMS_REQUEST_TYPE_IMS_COMMAND) specified for imsRequestType property is invalid.

User Action: ImsRequestType 2(IMS_REQUEST_TYPE_IMS_COMMAND) only valid with SYNC_SEND_RECEIVE interaction. ImsRequestType 1(IMS_REQUEST_TYPE_IMS_TRANSACTION) is required for SYNC_SEND, SYNC_END_CONVERSATION, SYNC_RECEIVE_ASYNCOUTPUT, SYNC_RECEIVE_ASYNCOUTPUT_SINGLE_NOWAIT and SYNC_RECEIVE_ASYNCOUTPUT_SINGLE_WAIT interactions.

ICO0119E

```
javax.resource.ResourceException:
ICO0119E: methodname error.
A supported SSL provider was not found. [caught_exception]
```

Explanation: When attempting to initialize a Secure Sockets Layer TCP/IP connection with IMS Connect, IMS Connector for Java needs to use one of the two supported providers, com.ibm.jsse.JSSEProvider or sun.security.provider.Sun. This error indicates that neither of these providers is available.

User Action: com.ibm.jsse.JSSEProvider should be added by default in an IBM JVM and sun.security.provider.Sun should be added by default in a Sun JVM. Ensure that you are running IMS Connector for Java in a supported IBM JVM if running in WebSphere Application Server or a Sun JVM in other application servers.

ICO0121E

```
javax.resource.ResourceException:
ICO0121E: methodname error.
Invalid reRoute name value. Prefix HWS is reserved for use by
IMS Connector for Java.
```

Explanation: The value specified for reRouteName property is invalid. The prefix 'HWS' is reserved for use by IMS Connector for Java.

User Action: Provide a valid value for reRouteName property. A valid value should adhere to the following rules:

- Is not a null string
- Does not start with a blank field
- Does not start with the IMS Connector for Java reserved prefix 'HWS'
- Is 8 characters long
- Uses the valid characters A - Z, 0 - 9, @, #, and $

ICO0122E

```
javax.resource.ResourceException:
ICO0122E: methodname error.
Invalid reRoute value. When purgeAsyncOutput value is true, reRoute
value cannot be true.
```

Explanation: The value specified for reRoute property is invalid. Because the value specified for purgeAsyncOutput property is TRUE, or the default value (TRUE) is used for purgeAsyncOutput property.

User Action: Ensure to set purgeAsyncOutput property to FALSE, if you want to set reRoute to TRUE.

Notices

The XDoclet Documentation included in this IBM product is used with permission and is covered under the following copyright attribution statement: Copyright (c) 2000-2004, XDoclet Team. All rights reserved.

Portions based on *Design Patterns: Elements of Reusable Object-Oriented Software*, by Erich Gamma, Richard Helm, Ralph Johnson and John Vlissides, Copyright (c) 1995 by Addison-Wesley Publishing Company, Inc. All rights reserved.

U.S. Government Users Restricted Rights - Use, duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp.

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 documentation 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 documentation. The furnishing of this documentation 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 OR CONDITIONS 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.

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:

Intellectual Property Dept. for Rational Software
IBM Corporation
20 Maguire Road
Lexington, Massachusetts 02421-3112
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 documentation 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 contains examples of data and reports used in daily business operations. To illustrate them as completely as possible, the examples may 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 illustrates 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. 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 IBM's application programming interfaces.

Each copy or any portion of these sample programs or any derivative work, must include a copyright notice as follows:

(C) (your company name) (year). Portions of this code are derived from IBM Corp. Sample Programs. (C) Copyright IBM Corp. 2000, 2005. All rights reserved.

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

Programming interface information

Programming interface information 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

See http://www.ibm.com/legal/copytrade.shtml.

Printed in USA

SC18-9593-01