iWay Software
An Information Builders Company

iWay

iWay Connector for J2EE Connector
Architecture User's Guide
Version 5 Release 5

Updated for J2EE CA 1.5

DN3501521.0305

Preface

This document is for iWay customers, consultants, and resellers who integrate J2EE™ application components with J2EE application servers using the iWay Connector for JCA (JCA). Before you begin, you should understand the J2EE Connector Architecture Specification, JSR016.

This documentation describes how to use the iWay Connector for JCA. It is intended for those who are using the J2EE Connector Architecture to connect to Enterprise Information Systems (EIS).

How This Manual Is Organized

The following table lists the titles and numbers of the chapters and appendix for this manual with a brief description of the contents of each chapter or appendix.

Chapter/Appendix		Contents
1	Introducing the iWay Connector for JCA	Introduces the iWay Connector for JCA and provides information on how the connector is distributed and deployed.
2	Deploying the iWay Connector for JCA	Describes how to configure and deploy the iWay Connector for JCA.
3	iWay JCA Installation Verification Program	Describes the iWay JCA Installation Verification Program, used to test the functionality of the iWay Connector for JCA in the iWay adapter framework.
A	Servlet Sample Code	Contains servlet sample code for Enterprise JavaBeans™.

Documentation Conventions

The following table lists and describes the conventions that apply in this manual.

Convention	Description
`THIS TYPEFACE` or `this typeface`	Denotes syntax that you must enter exactly as shown.
`this typeface`	Represents a placeholder (or variable) in syntax for a value that you or the system must supply.
`underscore`	Indicates a default setting.
this typeface	Represents a placeholder (or variable), a cross-reference, or an important term.

this typeface	Highlights a file name or command.
Key + Key	Indicates keys that you must press simultaneously.
{ }	Indicates two or three choices; type one of them, not the braces.
|	Separates mutually exclusive choices in syntax. Type one of them, not the symbol.
. . .	Indicates that you can enter a parameter multiple times. Type only the parameter, not the ellipsis points (…).
. . .	Indicates that there are (or could be) intervening or additional commands.

Related Publications

Visit our World Wide Web site, http://www.iwaysoftware.com, to view a current listing of our publications and to place an order. You can also contact the Publications Order Department at (800) 969-4636.

For information on installing the iWay Connector for J2EE Architecture, see the *iWay Installation and Configuration* manual.

Customer Support

Do you have questions about the iWay Connector for JCA?

If you bought the product from a vendor other than iWay Software, contact your distributor.

If you bought the product directly from iWay Software, call Information Builders Customer Support Services (CSS) at (800) 736-6130 or (212) 736-6130. Customer Support Consultants are available Monday through Friday between 8:00 A.M. and 8:00 P.M. EST to address all your iWay questions. Our consultants can also give you general guidance regarding product capabilities and documentation. Please be ready to provide your six-digit site code number (*xxxx.xx*) when you call.

You can also access support services electronically, 24 hours a day, with InfoResponse Online. InfoResponse Online is accessible through our World Wide Web site, http://www.iwaysoftware.com. It connects you to the tracking system and known-problem database at our support center. Registered users can open, update, and view the status of cases in the tracking system and read descriptions of reported software issues. New users can register immediately for this service. The technical support section of www.informationbuilders.com also provides usage techniques, diagnostic tips, and answers to frequently asked questions.

To learn about the full range of available support services, ask your iWay Software representative about InfoResponse Online, or call (800) 969-INFO.

Help Us to Serve You Better

To help our consultants answer your questions effectively, please be prepared to provide specifications and sample files and to answer questions about errors and problems.

The following tables list the specifications our consultants require.

Platform	
Operating System	
OS Version	
Product List	
Adapters	
Adapter Deployment	For example, JCA, Business Services Engine, iWay Adapter Manager
Container Version	

The following table lists components. Specify the version in the column provided.

Component	Version
iWay Adapter	
EIS (DBMS/APP)	
HOTFIX / Service Pack	

The following table lists the types of Application Explorer. Specify the version (and platform, if different than listed previously) in the columns provided.

Application Explorer Type	**Version**	**Platform**
Swing		
Servlet		
ASP		

In the following table, specify the JVM version and vendor in the columns provided.

Version	**Vendor**

The following table lists additional questions to help us serve you better.

Request/Question	**Error/Problem Details or Information**
Provide usage scenarios or summarize the application that produces the problem.	
Did this happen previously?	
Can you reproduce this problem consistently?	
Any **change in the application environment**: software configuration, EIS/ database configuration, application, and so forth?	
Under what circumstance does the problem *not* occur?	
Describe the **steps** to reproduce the problem.	
Describe the **problem**.	
Specify the **error** message(s).	

The following table lists error/problem files that might be applicable.

XML schema
XML instances
Other input documents (transformation)
Error screen shots
Error output files
Trace and log files
Log transaction

User Feedback

In an effort to produce effective documentation, the Documentation Services staff welcomes your opinions regarding this manual. Please use the Reader Comments form at the end of this manual to communicate suggestions for improving this publication or to alert us to corrections. You also can go to our Web site, http://www.iwaysoftware.com and use the Documentation Feedback form.

Thank you, in advance, for your comments.

iWay Software Training and Professional Services

Interested in training? Our Education Department offers a wide variety of training courses for iWay Software and other Information Builders products.

For information on course descriptions, locations, and dates, or to register for classes, visit our World Wide Web site, http://www.iwaysoftware.com or call (800) 969-INFO to speak to an Education Representative.

Interested in technical assistance for your implementation? Our Professional Services department provides expert design, systems architecture, implementation, and project management services for all your business integration projects. For information, visit our World Wide Web site, http://www.iwaysoftware.com.

Contents

1. Introducing the iWay Connector for JCA 1-1

Overview of the iWay Connector for JCA 1-2

Support for Over 250 Adapters 1-3

Flexibility of Use 1-4

Distribution of the iWay Connector for JCA 1-6

Deployment of the iWay Connector for JCA 1-7

Using a JCA Resource Adapter 1-9

Altering JCA Resource Adapter Connection Properties 1-13

New in the iWay Connector for JCA Version 1.5 1-13

Using Programs Written for iWay Connector for 1.0 with the New iWay Connector for JCA 1.5 . 1-13

2. Deploying the iWay Connector for JCA 2-1

Deploying to Sun Java System Application Server 2-2

3. iWay JCA Installation Verification Program 3-1

Overview of the IVP 3-2

Deploying and Running the IVP for Sun Java System Application Server 3-2

Deploying the JCA Test Tool 3-3

Running the JCA Test Tool 3-4

Configuring the JCA Test Tool 3-5

4. Configuring and Deploying the iWay JCA 1.5 Sample Applications 4-1

The iWay JCA 1.5 Connector Architecture 4-2

The iWay JCA 1.5 Sample Applications 4-2

The MessageListener Interface 4-4

Configuring and Deploying the Sample Applications 4-4

A. Servlet Sample Code A-1

iWay Servlet Sample Code A-2

CHAPTER 1

Introducing the iWay Connector for JCA

Topics:

- Overview of the iWay Connector for JCA
- Distribution of the iWay Connector for JCA
- Deployment of the iWay Connector for JCA
- Using a JCA Resource Adapter

This section provides an overview of the iWay Connector for J2EE Connector Architecture (JCA) and describes how it is distributed and deployed.

Overview of the iWay Connector for JCA

The J2EE Connector Architecture (JCA) defines a standard architecture for connecting the J2EE platform to a heterogeneous Enterprise Information System (EIS). Examples of an EIS include Enterprise Resource Planning (ERP), Customer Relationship Management (CRM), Supply Chain Management (SCM), mainframe transaction processing, database systems, and legacy applications that are not written in the Java™ programming language. By defining a set of scalable, secure, and transactional mechanisms, JCA enables the integration of an EIS with an application server and enterprise applications.

The J2EE Connector Architecture permits an EIS vendor to provide a standard resource adapter for its EIS. The resource adapter plugs into an application server, providing connectivity to an EIS, and integrating it with the rest of the enterprise. If an application server vendor has extended its system to support JCA, it is assured of seamless connectivity to multiple Enterprise Information Systems.

iWay Software is the world leader in providing integration tools and adapter technologies that offer fluid control over all your enterprise information assets. The iWay set of standards-based adapters provide rapid integration of applications, e-business documents, and databases. The iWay Intelligent Adapter Suite integrates a wide variety of IT assets into a single virtual information system with shared data structures, transactions, and business logic. No matter how diverse or dissimilar the components of your IT infrastructure are, iWay integration tools and adapters make them all accessible for e-business applications and integration projects.

Companies have adopted J2EE application servers as a standard framework for supporting the development and maintenance of Internet applications. Separating non-functional tasks such as security, transactions, connection pooling, and data persistence from functional tasks such as order processing and customer value analysis enables you to concentrate on developing business logic and software vendors to focus on application infrastructure.

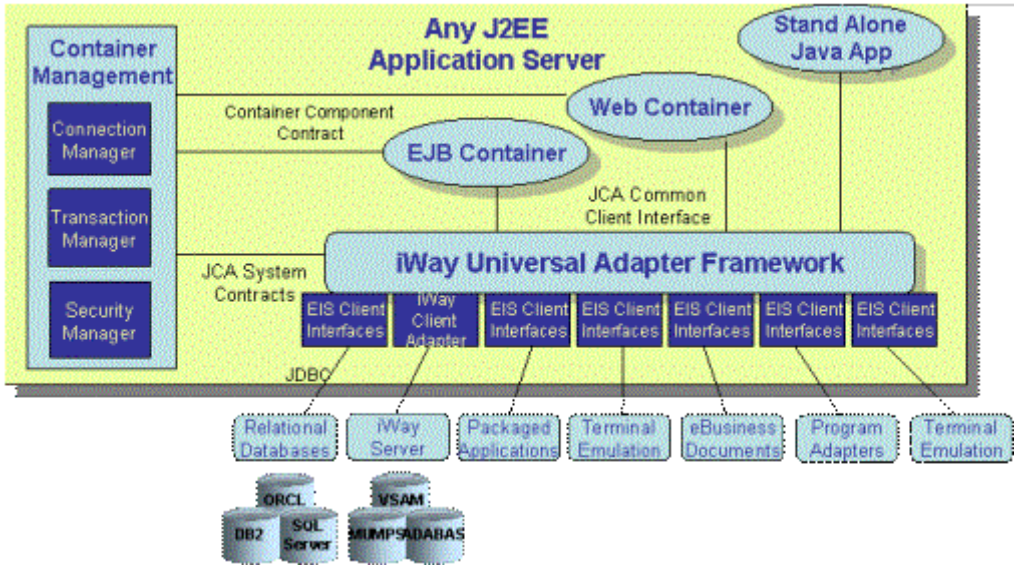
The JCA specification was created to establish a standardized mechanism for integrating Java-based applications with heterogeneous enterprise information assets such as packaged systems. JCA provides a standard way to connect several application servers with many Enterprise Information Systems. Using the iWay Connector for J2EE Connector Architecture (iWay Connector for JCA), you can connect to any of over 250 adapters.

Support for Over 250 Adapters

The iWay Connector for J2EE Connector Architecture (iWay Connector for JCA), provides over 250 intelligent adapters, enabling you to integrate your J2EE applications with your Enterprise Information Systems, all within a unified adapter framework. iWay provides intelligent adapters for:

- Application packages, such as Siebel, SAP, PeopleSoft, Ariba, Lawson, Clarify, and others.
- Electronic business formats, such as SWIFT, FIX, HIPAA, ACORD, ebXML, and OAG BODs.
- Legacy database systems, such as ADABAS, Ingres, MUMPS, Teradata, Unisys, and others.
- Relational databases, such as Oracle, MS SQL Server, Informix, Sybase, and many more.
- Transaction environments, such as CICS, IMS, and Tuxedo.
- 35 mainframe, midrange, UNIX, and PC operating platforms.
- Custom 3GL and 4GL applications.
- .NET Assemblies
- Messaging products like Sonic MQ, Oracle AQ, MQSeries, or JMS.
- Object technologies like COM, CORBA, and EJB™.
- Terminal emulation adapters that provide wrappers around the presentation tiers of applications of "green screen" 3270 and 5250 systems.

The following diagram shows the iWay universal adapter framework as deployed to a standard J2EE application server environment. On the left is a large rectangle representing container management. Inside container management are three smaller rectangles, representing connection manager, transaction manager, and security manager. Container management connects to the EJB container and the Web container (represented by ovals) through the container component contract. The EJB container, the Web container, and the stand-alone Java app (represented by an oval) connect to the iWay Universal Adapter Framework through the JCA common client interface. The container management also connects to the iWay universal adapter framework through the JCA system contracts. The iWay Universal Adapter Framework includes the iWay client adapter (that connects to the iWay server) and the EIS client interfaces (that connect through JDBC to relational databases, packaged applications, terminal emulators, eBusiness documents, and program adapters).



Flexibility of Use

Bidirectional Outbound/Inbound Use. You can use iWay adapters to invoke traditional response-request remote procedure calls (RPC) to your Enterprise Information Systems. This includes SQL requests to relational and non-relational data stores, as well as calls to applications such as SAP or Siebel. Additionally, iWay adapters work with many messaging technologies and have a native Java Message Service (JMS) interface. This enables you to augment your asynchronous messaging systems with iWay-enabled systems access.

Service Adapters. iWay resource adapters can invoke services on their targeted EIS.

Event Listener. iWay adapters can listen for events occurring on their targeted application server. For example, a new Siebel customer entry can trigger an event that causes the adapter to invoke Enterprise JavaBeans (EJB) within your application server, which in turn can invoke a service on CICS and SAP.

The iWay Connector for JCA can be used as a stand-alone application, or in conjunction with a supported application server such as a Sun™ J2EE Application Server, Oracle 9i Application Server Containers for Java, BEA WebLogic Server, Fujitsu Interstage Application Server, IBM WebSphere Application Server, or the Novell exteNd Application Server.

This documentation is not intended as a primer on JCA itself, except as required to illustrate the iWay Connector for JCA. For more information about JCA, see the *J2EE Connector Architecture Specification*, JSR016 (Java Community Process), Sun Microsystems, July 25, 2001.

iWay packages two connectors: one that conforms to the J2EE Connector Architecture (JCA) Specification version 1.0. and one that conforms to the JCA Specification version 1.5. These specifications approach the definition of a standard for application interaction with an Enterprise Information System (EIS). Before JCA, most EIS vendors offered vendor-specific architectures to provide connectivity between applications and their software; each program interacting with an EIS was required to be hand-tooled with a detailed knowledge of the peculiarities of the EIS.

Although common APIs such as ODBC and JDBC™ address application interaction with SQL data sources, the heterogeneous nature of EIS makes such APIs unwieldy. iWay Software provides a common interface to underlying EIS adapters. An access API that meets the requirements of both relational and packaged application sources required. JCA defines standard Java interfaces for simplifying the integration of applications with the EIS.

JCA addresses connection management, transaction management, and security. It includes the following components:

- Application server
- Resource adapter
- Application

The resource adapter represents the interests of the underlying EIS. The application server is not strictly required. The application interacts with the resource adapter using what JCA calls standard contracts. Standard contracts define what interactions are to take place and how they appear. The contract between the application and the resource adapter is called the Common Client Interface (CCI). The resource adapter, in turn, interacts with the application manager under the Service Provider Interface (SPI), defining how the management of the resource adapter occurs. This includes:

- Connectivity management
- Transaction demarcation
- Event listening (listeners receive notification of significant events; for example, a connection failure)
- Pooling of connections and other resources

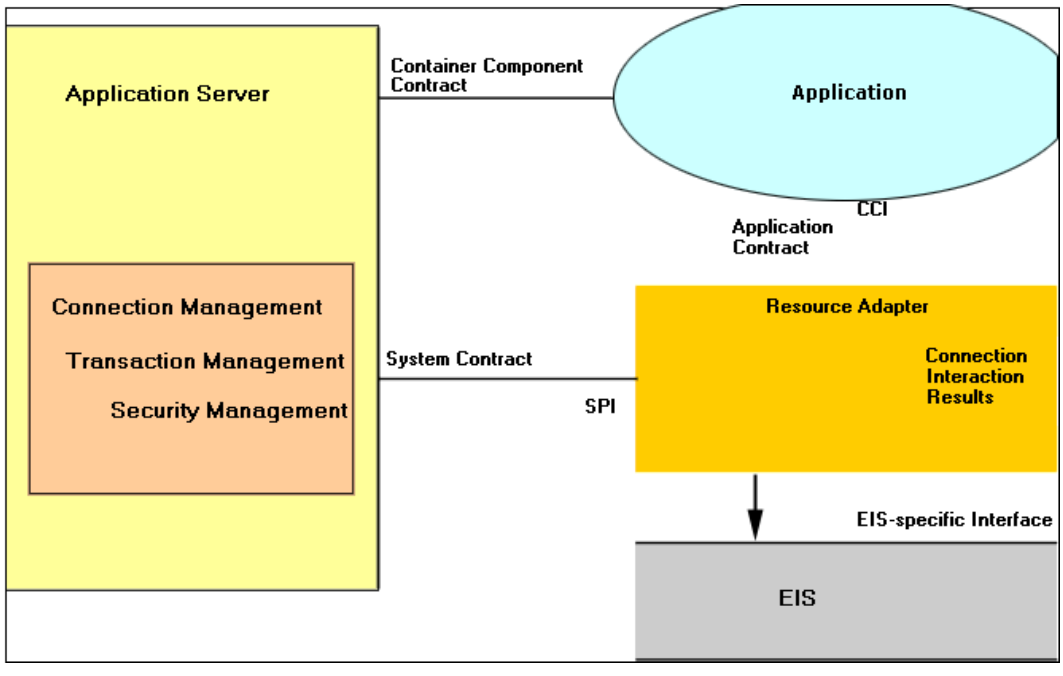
In the normal course of events, the application server manages the transaction. First, the application uses a naming service to locate the appropriate resource adapter. The application server supplies the naming service, and so it recognizes that a request is being made to locate a resource adapter. In such a case, the application server interposes an intermediate object supplied by the resource adapter that interacts between the resource adapter and the application server. Through this intermediating object, the application server manages the items within the SPI contract below the awareness of the ultimate application.

While the application uses the resource adapter, the application server manages the activity to the side, however, it appears to the application that it is interacting directly with the resource adapter.

Distribution of the iWay Connector for JCA

The iWay Connector for JCA 1.0 and the iWay Connector for JCA 1.5 are J2EE Connector Architecture resource adapters. They are distributed as both a standard Resource Adapter Archive (RAR) for deployment to the application server and as a JAR file for stand-alone application use. Thus, the connector can be employed in systems that are non-compliant, although services such as pooled connections are not available.

The following diagram shows the interaction between a JCA resource adapter (represented by a gold rectangle on the right side in the middle that connects to an EIS represented by a gray rectangle at the bottom right) and a standard application server (represented by a yellow rectangle on the left that contains a light orange rectangle that represents connection management, transaction management, and security management). The application server connects to the application (represented by a blue oval) through the container component contract. The application server also connects to the resource adapter through the system contract.



Deployment of the iWay Connector for JCA

The iWay Connector for JCA is a JCA resource adapter. For the connector to operate with an Enterprise JavaBean (EJB), it must be deployed to a Web application server. The RAR file is identified to the application server, which then uses the file as any other component. The connector name is registered with the name service (JNDI) to enable the EJB (or servlet) to locate and instantiate it for use.

iWay's universal resource adapter framework can be used to access any adapter in the iWay repository. This framework avoids the need to deploy one resource adapter per EIS/back-end system. At run time, a CCI program can access any of the adapters in the repository by passing an adapter and target name through a connection factory and connection specification.

At deploy time, the user sets the default adapter name and target name. At run time, a CCI call is made with the connection specification to obtain the connection. If the specific adapter name and target name are passed in the connection specification, they are used for obtaining the connection. However, if the adapter name and target name are not passed in the connection specification, the default adapter and the default target are used for obtaining the connection.The following CCI calls can be used to establish a connection:

```
getConnection()
```

or

```
getConnection(ConnectionSpec)
```

The first relies only on the ManagedConnectionFactory (MCF) defaults. The second is the AdapterName and Target if provided in the ConnectionSpec.

You can control the adapter in the following ways:

- Specify default AdapterName and Target parameters at deploy time.

  If these are left blank and an AdapterName and Target are not provided in the connection specification, run time errors occur.

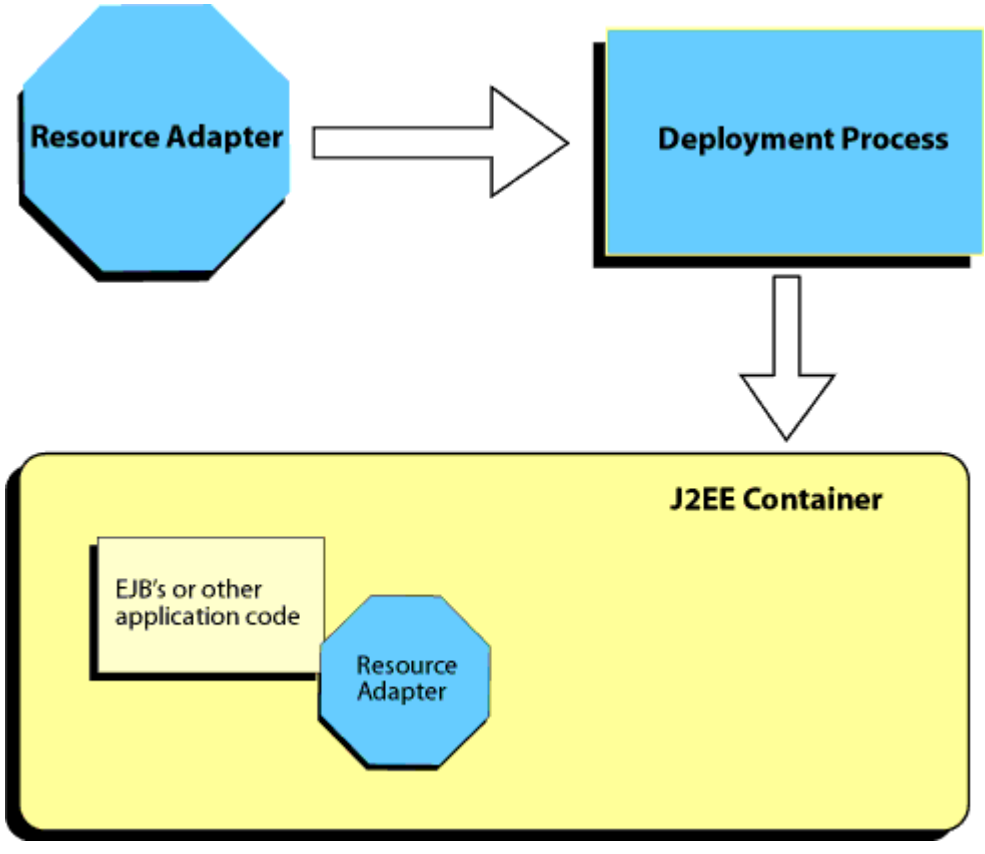
  In this case, the default values can be overridden at run time.

- The AdapterName and Target parameters are in your MCF.

  The AdapterName and Target parameters are used only when creating a connection pool. You can have many pools, each with a different AdapterName and Target, all using the same Resource Adapter that was deployed only once.

  A CCI application uses the JNDI for the desired connection pool and calls the getConnection() method. Of course, the CCI program could still overwrite AdapterName and Target using the getConnection(ConnectionSpec) method.

The following diagram shows the deployment of a JCA resource adapter (represented by a blue hexagram on the upper left with an arrow directed towards the deployment process). The deployment process is represented by a blue rectangle on the upper right which in turn has an arrow directed downward toward a yellow rectangle which represents the J2EE container. Inside the J2EE container is a rectangle representing EJBs or other application code and a blue hexagram representing the resource adapter.



Using a JCA Resource Adapter

Using any JCA resource adapter (such as the iWay Connector for JCA) is, as of the 1.5 specification, a programming effort as well as an assembly effort. A JCA resource adapter appears to a programmer as two interacting parts. You can configure and serialize specification components with standard bean tools. These are specific to the adapter and require the programmer to understand the configuration properties that they offer.

Contract components meet stricter interface requirements and can be used by the JCA-compliant application exactly as described in the specification.

Both the 1.0 and the 1.5 adapters provide specification and contract components: the application programmer can write generic code to assemble the specification and contract components, and thus interact with the underlying Enterprise Information Systems (EIS). However, the tasks of preparing input and understanding output remain EIS-specific. The connector exposes the following iWay-specific components to the JCA application.

- **IWAFConnectionSpec (for JCA 1.0 and JCA 1.5 connectors).** A JavaBean encapsulating the properties required to perform a connection to the iWay service adapters.

  You can use any standard bean tool to set the properties and serialize the bean.

  Properties offered by the iWay Connector for JCA connection specification are those required to control a local invocation of the iWay packaged adapters.

  The IWAFConnectionSpec for the **JCA 1.0 connector** has seven parameters:

  - Adapter name
  - Adapter configuration name
  - Language
  - Country
  - User name
  - Password
  - Log level

  The IWAFConnectionSpec for the **JCA 1.5 connector** has five parameters:

  - Adapter name
  - Target
  - User name
  - Password
  - Log level

  The IWAFConnectionSpec supports connection pooling based on the previous configuration parameters.

- **IWAFInboundConnectionSpec (for JCA 1.0 connector only).** A JavaBean encapsulating the properties required to perform a connection to the iWay Event adapter channel.

  You can use any standard bean tool to set the properties and serialize the bean.

Properties offered by the iWay Connector for JCA connection specification are those required to control a local invocation of the iWay adapters. The IWAFInboundConnectionSpec has the following parameters:

- Adapter name
- Language
- Country
- Channel
- Disposition
- Log level

- **IWAFInteractionSpec (for JCA 1.0 and JCA 1.5 connectors).** A JavaBean encapsulating the properties required to manage one interaction with the adapter.

  As is the case with the IWAFConnectionSpec, you can use this object with any bean tool.

  The IWAFInteractionSpec uses the standard IndexedRecord to return the output.

- **IWAFInboundInteractionSpec (JCA 1.0 connector) and IWAFActivationSpec (JCA 1.5 connector).** A JavaBean encapsulating the properties required to start and stop the iWay channel.

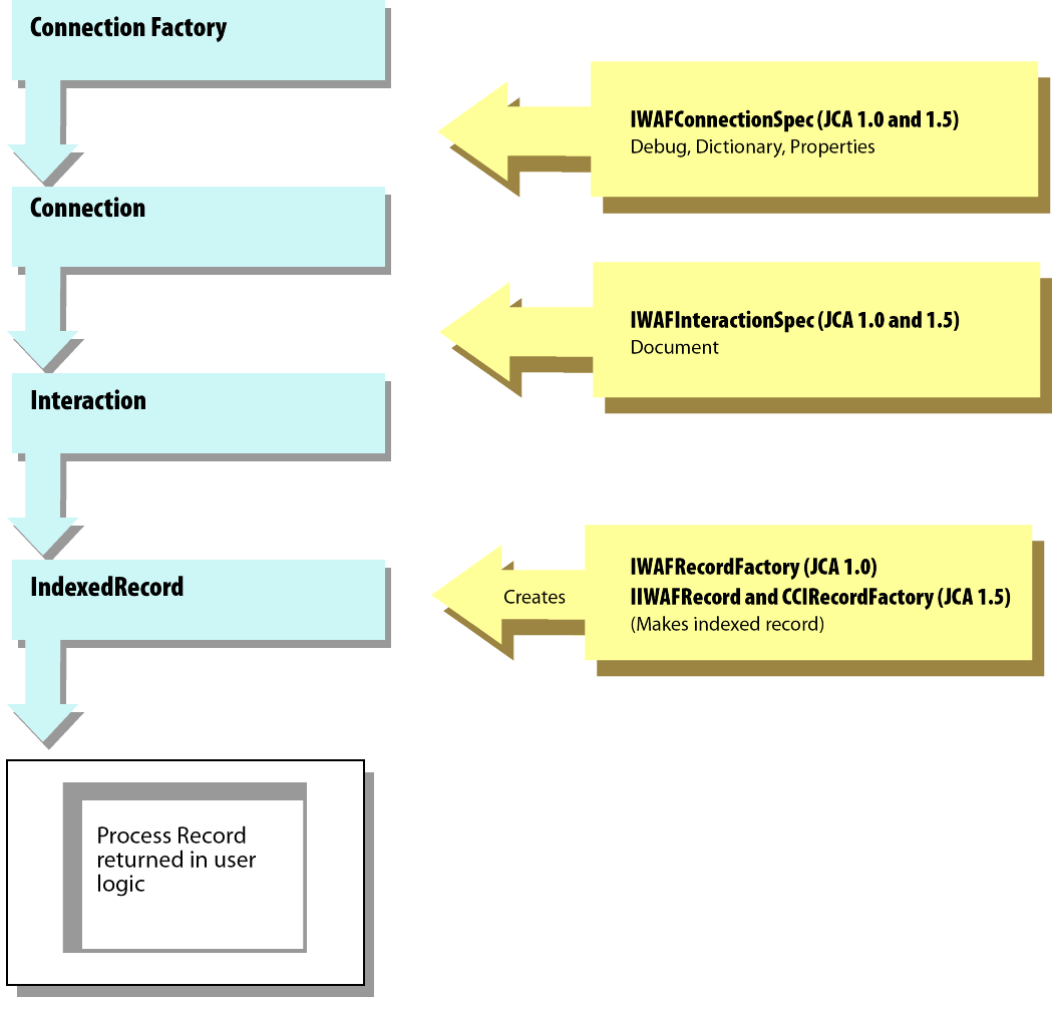
  As is the case with the IWAFInbound ConnectionSpec, you can use this object with any bean tool.

- **IWAFRecordFactory (JCA 1.0 connector).** An object used to construct records to be passed between the application and the adapter at design time.

- **IIWAFRecord and CCIRecordFactory (JCA 1.5 connector).** Objects used to construct records to be passed between the application and the adapter at design time. CCIRecordFactory is iWay's implementation of the RecordFactory interface. It allows the creating of IIWAFRecords besides Indexed and Mapped records

The following diagram shows the relationships between the components in a simple stand-alone application. In such an application, transactions issues and security are ignored. When an application uses the connector within a hosted application server (such as Sun Java System Application Server), it is referred to as a *managed* application. From the programmer's viewpoint, the most significant difference is the use of the Java JNDI name search facility to locate the connection factory.

After the connection factory is located and instantiated, the application control flow appears much the same. The connector supports non-transactional and locally controlled transactional workflows, with coordination by the Application Server.



Altering JCA Resource Adapter Connection Properties

The JCA Resource Connector has an initial capacity value of 0 by default and cannot be changed. The maximum capacity value is 10 by default and can be changed to a higher value.

New in the iWay Connector for JCA Version 1.5

iWay has updated the iWay Connector for JCA to support the JCA version 1.5 specification. The following new features are suppoted in accordance with the JCA 1.5 specification:

- Inbound messaging
- Lifecycle management contract
- Work management contract
- Message inflow contract
- JCA 1.5 packaging

For more information on writing CCI programs that take advantage of these features, see the Sun specification:

`http://java.sun.com/j2ee/connector/`

Using Programs Written for iWay Connector for 1.0 with the New iWay Connector for JCA 1.5

Due to the changes in the JCA specification, some of the APIs names have changed from the iWay Connector for JCA 1.0 to the iWay Connector for JCA 1.5. You should review these changes to ensure that a CCI program written to interact with the iWay Connector for 1.0 will work with the iWay Connector for JCA 1.5.

To review the API name changes, review the JAVA docs installed in the iWay installation directory, for example,

`D:\Program Files\iWay55\etc\doc`

The file is called iwjca15-javadoc.zip. Extract the contents to a directory of your choice and open the index.html file with a browser.

CHAPTER 2

Deploying the iWay Connector for JCA

Topic:

- Deploying to Sun Java System Application Server

The following topic describes how to deploy the iWay Connector for J2EE Architecture (JCA) to Sun Java System Application Server.

Deploying to Sun Java System Application Server

Before deploying the iWay Connector for JCA to Sun Java System Application Server, you must configure security settings and endorsed files. If you use the iWay Connector for JCA, disable server.policy as explained in the following procedure.

iWay components relay on several files and classes that must be loaded before any of your application server default classes. To ensure these classes are loaded in the correct order, see *How to How to Configure Endorsed Files on page 2-2*.

Procedure How to Configure Endorsed Files

iWay components rely on several files and classes that must be loaded before any of your application server default classes.

To ensure classes load in the correct order:

1. Determine your application server JAVA_HOME directory. If you installed a Java SDK/JDK with your application server, the default location on Windows is usually:

   ```
   C:\SUN\AppServer\jdk
   ```

   On other platforms, use the corresponding location.

   If you specified a different Java installation, you can determine JAVA_HOME by looking at the asenv.bat or asenv.conf file for your applications server. For the default domain and server on Windows, this is:

   ```
   C:\Sun\AppServer\config\asenv.bat
   ```

   On other platforms, use the corresponding location.

2. If it does not exist, create an endorsed directory in the jre/lib or jdk/lib directory, for example:

   ```
   C:\SUN\AppServer\jdk\lib\endorsed
   ```

   or

   ```
   C:\j2sdk1.4.2_03\jre\lib\endorsed
   ```

3. Copy the iwafjca.rar file installed with iWay to the endorsed directory. The default location for iwafjca.rar on Windows is:

   ```
   C:\Program Files\iWay55\sun\iwafjca.rar
   ```

   On other platforms, use the corresponding location.

4. Open a command or shell prompt and navigate to the endorsed directory.

5. Use the JAR command to extract three files from iwafjca.rar.

   ```
   jar xvf iwafjca.rar xalan.jar xercesImpl.jar xmlParserAPIs.jar
   ```

If you receive an error, then the JAR command is probably not in your search path. You can add the JAR command to your search path or execute it using its full path. The jar command is located in the Java SDK bin directory, which varies depending on your Java release, for example:

```
C:\j2sdk1.4.2_03\bin\jar xvf iwafjca.rar xalan.jar xercesImpl.jar xmlParserAPIs.jar
```

6. Confirm that the endorsed directory now contains the following three files:

   ```
   xalan.jar
   xercesImpl.jar
   xmlParserAPIs.jar
   ```

7. Remove the iwafjca.rar file from the endorsed directory.
8. If your application server is running, restart it.

Procedure How to Configure Security Settings for iWay

You should disable server.policy as explained in the following procedure before deploying iWay components.

To configure security settings for iWay:

1. Using a text editor, open the domain.xml file for your domain. For the default domain on Windows, this is:

   ```
   C:\Sun\AppServer\domains\domain1\config\domain.xml
   ```

   On other platforms, use the corresponding location. For other domains, use the corresponding location.

2. Remove the following line:

   ```
   <jvm-options>-Djava.security.policy=${com.sun.aas.instanceRoot}/config
   /server.policy</jvm-options>
   ```

   It may appear multiple times and you can remove each appearance of the line.

3. Save and exit *domain.xml*.
4. If your application server is running, restart it.

Procedure: How to Deploy the iWay Connector for JCA

1. Start Sun Java System Application Server.
2. Log on to the Sun Admin Console. If you kept the default port, you can use:

   `https://hostname:4849`

   where:

   hostname

   Is the hostname for your application server.

   The Admin Console opens in your browser.
3. On the left, expand the *Applications* folder.
4. On the left, click *Connector Modules*.
5. On the right, click *Deploy*.
6. Specify the iwafjca.rar or iwjca15.rar file depending on which connector you use, for example:

   `C:\Program Files\iWay55\sun\iwjca15.rar`

   or

   `C:\Program Files\iWay55\sun\iwafjca.rar`
7. Click *Next*.
8. Ensure the *Application Name* is one of the following depending on which connector you deploy:

   `iwjca15`

   or

   `iwafjca`
9. If you are using the JCA 1.5 connector, under Resource Adapter Properties, provide values for the names listed in the following table:

Name	Value
IWayConfig	The configuration name. The default is: `base`
IWayRepoURL	Clear this field so it has no value.

Name	Value
LogLevel	The log level, for example: DEBUG
IWayRepoPassword	Clear this field so it has no value.
IWayRepoUser	Clear this field so it has no value.
IWayHome	Directory where iWay is installed, for example: C:\Program Files\iWay55

10. Specify which servers you are deploying into by moving them from the Available to Selected boxes.
11. Click *OK*.

    When the connector is deployed, it is expanded to a directory under the application server, for example:

    ```
    C:\Sun\AppServer\domains\domain1\applications\j2ee-modules\iwjca15
    ```

    or

    ```
    C:\Sun\AppServer\domains\domain1\applications\j2ee-modules\iwafjca
    ```

    If you receive an error, stop the server instance and remove each appearance of the following line from domain.xml file for your domain (AppServer\domains\domain1\config\domain.xml):

    ```
    <jvm-options>-Djava.security.policy=${com.sun.aas.instanceRoot}/config
    /server.policy</jvm-options>
    ```

Procedure How to Create Connection Pools for the iWay Connector for JCA 1.5

For JCA 1.5, to create a connection pool for the connector:

1. On the left of the Admin Console, expand *Resources*.
2. Under *Resources*, expand *Connectors*.
3. Click *Connector Connection Pools*.
4. On the right, click *New*.
   a. In the Name field type:

      ```
      eis/iWay
      ```

   b. For the Resource Adapter menu, choose:

      ```
      iwjca15
      ```

5. Click *Next*.
6. Click *Next* again.
7. From the Transaction Support drop-down menu, choose one of the following:

   `NoTransaction`

   or

   `LocalTransaction`

   Note: Leave the *Additional Properties* as they appear.
8. Click *Finish*.
   a. On the left, under *Connectors*, click *Connector Resources*.
   b. On the right, click *New*.
   c. In the JNDI Name field enter:

      `eis/iWay`

   d. For the Pool Name menu, choose:

      `eis/iWay`

   e. In the Targets area, select your sever or servers in the Available text box and click *Add*.
9. Click *OK*.

CHAPTER 3

iWay JCA Installation Verification Program

Topics:

- Overview of the IVP
- Deploying and Running the IVP for Sun Java System Application Server

The iWay JCA Installation Verification Program (IVP) is a JSP™-based test tool for interacting with iWay adapters. There is one test tool for the iWay Connector for J2EE Connector Architecture (iWay Connector for JCA) version 1.0 and one for the iWay Connector for JCA 1.5.

This test tool is used to test the functionality of the iWay Connector in the iWay adapter framework. There are several types of adapters available through the connector.

For more information on installing and configuring the JCA test tool, see the *iWay Installation and Configuration* manual.

Overview of the IVP

The iWay JCA Installation Verification Program (IVP):

- Supports the execution of iWay Service requests.
- Monitors of iWay JCA events.
- Determines which iWay adapters are installed.
- Reads the iWay JCA configuration repository.
- Loads the configurations for each configured adapter.
- Executes iWay Service requests for a given adapter.

The IVP provides tools that enable you to test application performance early in the development cycle. This allows enough time to make architectural changes and implementation changes, reducing risk early in the cycle, and avoiding problems in final performance tests.

The iWay Connector for J2EE Architecture includes deployable code and sample files. The sample files help you integrate the iWay JCA solution into the J2EE application and then test it. The following topic describes how to deploy and use the IVP.

The topics describe how to:

- Deploy the IVP.
- Modify the IVP to use other configurations.
- Use the IVP to execute a service request.
- Use the IVP to monitor iWay events.

Deploying and Running the IVP for Sun Java System Application Server

The JCA Test Tool includes sample code that enables you to test iWay Service and Event Adapters.

For more information on installing and configuring the JCA Test Tool, see the *iWay Installation and Configuration* manual.

Deploying the JCA Test Tool

Deploy the JCA Test Tool to Sun Java System Application Server as explained in the following procedure.

Procedure: How to Deploy the JCA Test Tool to Sun Java System Application Server

You can use the Sun Admin Console to deploy the JCA Test Tool.

1. Start Sun Java System Application Server if it is not started.
2. Log on to the Sun Admin Console. If you kept the default port and access it locally, you can use:

   ```
   https://localhost:4849
   ```

   The Admin Console opens in your browser.
3. On the left, expand the *Applications* folder.
4. On the left, click *Web Applications*.
5. On the right, click *Deploy*.
6. Specify the iwjcaivp.war or iwjca15ivp.war file depending on which connector you use, for example:

   ```
   C:\Program Files\iWay55\sun\iwjca15ivp.war
   ```

   or

   ```
   C:\Program Files\iWay55\sun\iwjcaivp.war
   ```
7. Click *Next*.
8. Ensure the *Application Name* is one of the following depending on which connector you deploy:

   ```
   iwjca15ivp
   ```

   or

   ```
   iwjcaivp
   ```
9. Ensure the *Context Root* is one of the following depending on which connector you deploy:

   ```
   /iwjca15ivp
   ```

   or

   ```
   /iwjcaivp
   ```
10. Click *OK*.

When the Test Tool is deployed, it is expanded to a directory under the application server, for example:

```
C:\Sun\AppServer\domains\domain1\applications\j2ee-modules\iwjca15ivp
```

or

```
C:\Sun\AppServer\domains\domain1\applications\j2ee-modules\iwjcaivp
```

Running the JCA Test Tool

After deploying the JCA Test Tool, access it to test the deployment.

Procedure: How to Run the JCA Test Tool

To run the JCA Test Tool:

1. Open a browser to:

   ```
   http://hostname:port/iwjca15ivp
   ```

   or

   ```
   http://hostname:port/iwjcaivp
   ```

   Depending on which JCA Test Tool and connector you deployed.

   where:

   hostname

   Is the name of the machine where your application server is running.

   port

   Is the HTTP port for the application server.

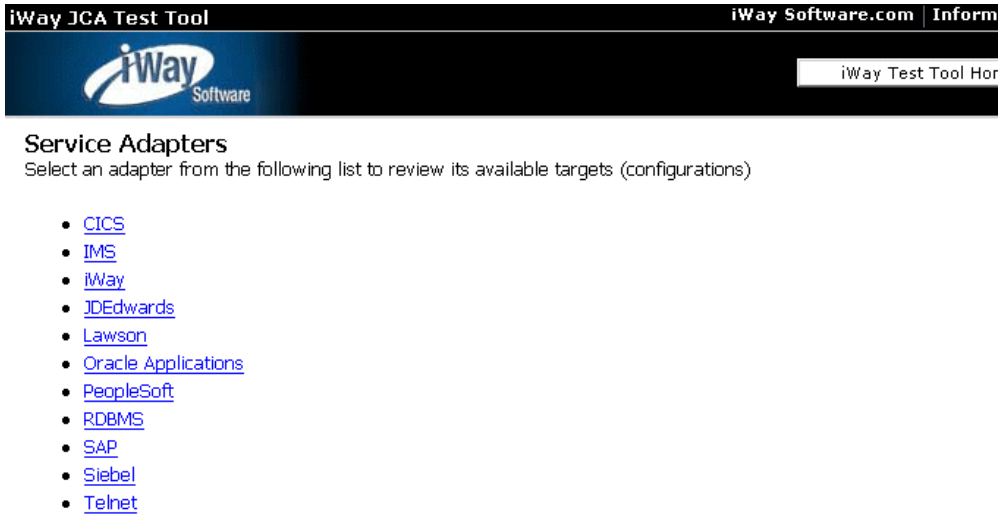
   For example:

   ```
   http://localhost:8080/iwjca15ivp
   ```

   The iWay JCA Test Tool window opens and provides a live list of iWay Service or Event adapters.

2. To display the available adapters, click *Service adapters* or *Event adapters*.

The following image shows a sample list of available adapters from which to choose.



The adapters that appear vary depending on the version of iWay you install and which files are in the iWay55\lib directory. If your adapter requires third party drivers or libraries, they must be in the lib directory or your adapter may not appear.

Initially, no targets are configured for the iWay Connector for JCA. However, after targets are configured using Application Explorer, you can test them using this tool.

After configuring targets using Application Explorer, you may need to restart the application server before the targets appear

Configuring the JCA Test Tool

The behavior of the JCA Test Tool is controlled by the following file inside the archive:

```
WEB-INF/web.xml
```

This file defines aspects of the JCA Test Tool running environment.

For JCA 1.5, there is no need to configure the test tool.

Procedure: How to Extract and Configure the JCA Test Tool

This is not a required configuration. It is provided for reference.

1. Extract the WEB-INF/web.xml file from the iwjcaivp.war or iwjca15ivp.war archive.

   a. Open a command prompt and navigate to the directory containing the Test Tool, for example:

   ```
   C:\Program Files\iWay55\sun
   ```

   b. Issue the following command:

   ```
   jar xvf iwjcaivp.war WEB-INF/web.xml
   ```

   or

   ```
   jar xvf iwjca15ivp.war WEB-INF/web.xml
   ```

   The JAR command is located in the Java SDK bin directory which might not be in your search path. If you receive an error, execute the JAR command using its full path. This path varies depending on which version of Java is installed, for example:

   ```
   C:\j2sdk1.4.1_03\bin\jar xvf iwjcaivp.war WEB-INF/web.xml
   ```

   Note: Be sure to use the JAR command and not Winzip. Winzip does not properly extract Java related archives.

2. Open the extracted web.xml file in a text editor.

3. Modify the contents of the <param-value> tags to change defaults. Ensure iway.home specifies the location of the iWay55 directory.

   Optionally, **provide the connection factory name for iWay Connector for JCA**. The connection factory is eis/IWAFConnectionFactory. The iWay Connector for JCAJCA Test Tool attempts to connect to the adapter via JNDI if this is defined. If this is undefined iway.home and iway.config are used instead.

   ```
   <context-param>
     <param-name>iway.jndi</param-name>
     <param-value></param-value>
     <description>
     JNDI name for the IWAF JCA Resource Adapter. If not
     provided, the application will create a new one based
     on iway.home, iway.config and iway.loglevel.
     </description>
   </context-param>
   ```

   For example:

   ```
   <param-value>eis/IWAFConnectionFactory</param-value>
   ```

Provide the directory where iWay 5.5. is installed by changing the path that appears.

```
<context-param>
  <param-name>iway.home</param-name>
  <param-value>c:\Program Files\iway55</param-value>
  <description>
  ONLY USED IF IWAY.JNDI NOT SET.
  Absolute path of iway installation directory.
  </description>
</context-param>
```

Optionally, **change the configuration to be used at run time**. A configuration named base is installed and available by default.

```
<context-param>
  <param-name>iway.config</param-name>
  <param-value>base</param-value>
  <description>
  ONLY USED IF IWAY.JNDI NOT SET.
  configuration name
  </description>
</context-param>
```

Optionally, **change the tracing level to debug, info, or error**.

```
<context-param>
  <param-name>iway.loglevel</param-name>
  <param-value>DEBUG</param-value>
  <description>
  ONLY USED IF IWAY.JNDI NOT SET.
  Log level: DEBUG FATAL ERROR INFO WARN
  </description>
</context-param>
```

4. Save and exit *web.xml*.
5. Use the JAR command to return the web.xml file to the WEB-INF directory within the archive.
   a. Ensure that you are in the following directory that contains the connector:

      ```
      C:\Program Files\iWay55\sun
      ```

   b. Issue the following command:

      ```
      jar uvf iwjcaivp.war WEB-INF/web.xml
      ```

      or

      ```
      jar uvf iwjca15ivp WEB-INF/web.xml
      ```
6. If the JCA Test Tool is deployed, undeploy it and then redeploy the edited version.

CHAPTER 4

Configuring and Deploying the iWay JCA 1.5 Sample Applications

Topics:

- The iWay JCA 1.5 Connector Architecture
- The iWay JCA 1.5 Sample Applications
- Configuring and Deploying the Sample Applications

This section describes the configuration, deployment, and use of the iWay J2EE CA version 1.5 (JCA 1.5) sample applications.

The iWay JCA 1.5 Connector Architecture

You can use the iWay universal resource adapter framework to access any adapter in the iWay repository. This framework eliminates the requirement to deploy one resource adapter per Enterprise Information System or back-end system. At run time, a CCI program can access any of the adapters in the repository by passing an adapter name and target name through a connection factory and connection specification.

At deploy time, you set the default adapter name and target name. At run time, a CCI call is made with the connection specification to establish the connection. If the specific adapter name and target name are passed in the connection specification, they are used for establishing the connection. If the adapter name and target name are not passed in the connection specification, the default adapter and the default target are used for establishing the connection.

The following CCI calls can be used to establish a connection:

```
getConnection()
```

or

```
getConnection(ConnectionSpec)
```

The first relies only on the ManagedConnectionFactory (MCF) defaults. The second is the AdapterName and Target if provided in the ConnectionSpec.

You can control the adapter in the following ways:

- Specify default AdapterName and Target parameters at deploy time.

  If these are left blank and an AdapterName and Target are not provided in the connection specification, run time errors occur.

  In this case, the default values can be overridden at run time.

- The AdapterName and Target parameters are in your MCF.

  The AdapterName and Target parameters are used only when creating a connection pool. You can have many pools, each with a different AdapterName and Target, all using the same Resource Adapter that was deployed only once.

  A CCI application uses the JNDI for the desired connection pool and calls the getConnection() method. Of course, the CCI program could still overwrite AdapterName and Target using the getConnection(ConnectionSpec) method.

The iWay JCA 1.5 Sample Applications

Two connectors are distributed in the iWay installation package. One conforms to the JCA 1.0 specification, with extensions that allow for the consumption of events. The other conforms to the JCA 1.5 specification.

The iWay installation package includes sample applications that demonstrate the use of the event and service capabilities of the iWay J2EE CA specification version 1.5 (JCA 1.5) connector. You can demonstrate the inbound and outbound capabilities of the iWay JCA 1.5 connector using the samples files.

Important: The iWay JCA 1.5 connector must be deployed before you configure and deploy the sample applications.

For information on in the J2EE CA 1.5 specification, see the following Web site:

```
http://java.sun.com/j2ee/connector/
```

The sample applications are packaged as follows:

- The inbound sample application is packaged as a JAR file, called

  ```
  iwjca15inbound.jar
  ```

- The outbound sample application is a Web application that is packaged as an EAR file, called

  ```
  iwjca15cci.ear
  ```

The JCA 1.0 connector provides event functionality through the configuration of ports and channels through Application Explorer. When using the adapter in conjunction with the iWay JCA 1.5 connector, you are not required to create event ports to dispose of event data. However, you must create a channel to enable event listening capabilities.

Instead of using ports configured in Application Explorer to dispose of event data, you create end points on the server that consume event data sent through a channel configured in Application Explorer. These end points are usually message-driven beans. End point activation or deactivation is performed by the application server.

The deployment descriptor file provided with the iWay JCA 1.5 connector, called ra.xml, specifies the classes of the connector that implement the required APIs as established by the J2EE-CA specification, version 1.5. The ra.xml file defines the activation configuration specification properties that must be provided by any message-driven bean that consumes iWay adapter events from the resource adapter. It also specifies the following listener interface that the message-driven beans must implement:

```
javax.resource.cci.MessageListener
```

This interface defines the following single method:

```
public javax.resource.cci.Record onMessage(javax.resource.cci.Record
record)
```

The MessageListener Interface

The sample message-driven bean that is provided implements the MessageListener interface. The interface is defined as follows in the descriptor file for the sample message-driven bean, called ejb-jar.xml:

```
<messaging-type>javax.resource.cci.MessageListener</messaging-type>
```

This matches the following inbound message listener type identified in the ra.xml file supplied with the iWay JCA 1.5 connector:

```
<messagelistener-type>javax.resource.cci.MessageListener</messagelistener
-type>
```

As noted in the Sun specification for JCA 1.5, to prepare the environment for end point activation and before any end point activations can occur, an end point deployer must configure properties that belong to the appropriate ActivationSpec JavaBean. The ActivationSpec JavaBean must be configured according to details in the end point deployment descriptor, as well as to message provider specifics.

Configuring and Deploying the Sample Applications

Before deploying the samples, ensure you have properly deployed and configured the iWay software for your application server. Before you deploy the sample applications, you also must use Application Explorer to create a channel for a particular adapter. The channel must be created while using JCA as the available host in Application Explorer.

For more information on installing and configuring iWay 5.5, see *iWay Installation and Configuration*. For information on creating channels in Application Explorer, see the user guide for your adapter.

To deploy the inbound sample application, you must first edit a descriptor file, called ejb-jar.xml. You edit the file to add the adapter and channel name for the adapter you test with the sample.

Important: Ensure you know the exact name of the adapter to test with the sample application and the exact name of the channel you configured for that adapter. If you edit the ejb-jar.xml file with values that do not match what you configured in Application Explorer, the deployment fails.

Procedure: How to Configure the Inbound Sample Application Settings

To configure the inbound sample application settings:

1. To change defaults, extract the *ejb-jar.xml* file from the iwjca15inbound.jar file.
   a. Open a command prompt and navigate to the directory containing the sample application, for example:

      ```
      C:\Program Files\iWay55\etc\setup\samples\jca15
      ```

b. Issue the following command:

```
jar xvf iwjca15inbound.jar META-INF/ejb-jar.xml
```

The JAR command is located in the Java SDK bin directory which might not be in your search path. If you receive an error, execute the JAR command using its full path. This path varies depending on which version of Java is installed, for example:

```
C:\j2sdk1.4.1_03\bin\jar xvf iwjca15inbound.jar META-INF
/ejb-jar.xml
```

Note: Ensure to use the JAR command and not Winzip. Winzip does not properly extract Java related archives.

2. Open the extracted ejb-jar.xml file in a text editor.
3. Modify the contents of the <activation-config-property-value> tags to change defaults for adapter name and channel name.

a. Ensure that a channel was configured in Application Explorer first and that you know the exact name of the channel and the adapter.

Deployment of the sample application fails if you use the wrong adapter name or channel name.

b. Change the following settings:

adapterName. The name of the adapter, for example, Siebel.

```
<activation-config-property>
<activation-config-property-name>adapterName</
activation-config-property-name>
<activation-config-property-value>Siebel
</activation-config-property-value>
</activation-config-property>
```

channelName. Name of the channel you have configured for the adapter identified in the adapterName property, for example, SiebelChannel.

```
<activation-config-property>
<activation-config-property-name>channelName
</activation-config-property-name>
<activation-config-property-value>SiebelChannel
</activation-config-property-value>
</activation-config-property>
```

4. Save the file and exit the editor.
5. Use the JAR command to return the ejb-jar.xml file to the META-INF directory within the archive.

a. Ensure that you are in the directory that contains the iwafjca15-samples-ejb.jar, for example:

```
C:\Program Files\iWay55\etc\setup
```

b. Issue the following command:

```
jar uvf iwafjca15inbound.jar META-INF/ejb-jar.xml
```

6. Deploy the connector.

For more information on deploying the connector, see *Deploy the Inbound Sample JCA 1.5 Application* on page 4-6

Procedure: How to Deploy the Inbound Sample JCA 1.5 Application

To deploy the inbound sample JCA 1.5 application:

1. Start Sun Java System Application Server.

2. Log on to the Sun Admin Console.

If you kept the default port and access it locally, you can use:

```
https://localhost:4849
```

The Admin Console opens in your browser.

a. On the left, expand the *Applications* folder.

b. On the left, click *EJB Modules*.

c. On the right, click *Deploy*.

d. Specify the *iwjca15inbound.jar* file, for example:

```
C:\Program Files\iWay55\etc\setup\iwjca15inbound.jar
```

3. Click *Next*.

4. Ensure the Application Name is:

```
iwafjca15inbound
```

5. Click *OK*.

The sample JCA 1.5 application deploys.

The application is expanded into a directory under your domain, for example:

```
C:\Sun\AppServer\domains\domain1\applications\j2ee-modules
\iwjca15inbound
```

After deployment, you can check the iWay Connector log and verify that the RA has started and received the information corresponding to the deployed message-driven bean. The channel specific logs are preceded by [adapterName_channel_channelName].

All log entries are preceded by the date, time, and thread ID. The following is an example of a log entry that follows a successful sample application deployment:

```
Wed, 23 Feb 2005 11:46:04.0384 EST - Thread[main,5,main] [debug]
------>
CCIInboundEventListener: onActivate()
Wed, 23 Feb 2005 11:46:04.0384 EST - Thread[main,5,main] [debug]
<------
CCIInboundEventListener: onActivate()
Wed, 23 Feb 2005 11:46:04.0384 EST - Thread[main,5,main] [info ]
[container] [Siebel.channel_SiebelChannel] pollThread started.
Wed, 23 Feb 2005 11:46:04.0384 EST - Thread[main,5,main] [info ]
<------
EndPointConsumer: start()
Wed, 23 Feb 2005 11:46:04.0384 EST - Thread[main,5,main] [debug]
Associated
adapterName:Siebel: channelName:SiebelChannel
com.iwaysoftware.afjca15.inflow.EndpointConsumer@1cdd76a
Wed, 23 Feb 2005 11:46:04.0384 EST - Thread[main,5,main] [debug]
<------
AbstractResourceAdapter: endpointActivation()
```

After the JCA 1.5 connector and sample application are deployed, an event sent by the iWay event adapter channel is consumed by the message-driven bean. When a message arrives at the message-driven bean, it acknowledges the receipt by replying to the sender. The way the message-driven bean reply is handled depends on the synchronization type of the iWay event adapter channel (REQUEST/REQUEST_RESPONSE/REQUEST_ACK).

The message-driven bean logs the receipt of the messages in the Application Server log file with entries similar to the following:

```
<MDB> In InboundMessageBean.InboundMessageBean()|#]
<MDB> In InboundMessageBean.setMessageDrivenContext()|#]
<MDB> In InboundMessageBean.ejbCreate()|#]
<MDB> ---- Got a message |#]
<MDB> XML DATA: <Siebel>
<productLine name="Siebel">
<token name="PA52">
<timestamp>20021107153406</timestamp>
<record>
<Company>60</Company>
```

Procedure: How to Deploy the Outbound Sample JCA 1.5 Application

To deploy the outbound sample JCA 1.5 application:

1. Start Sun Java System Application Server.
2. Log on to the Sun Admin Console.

   If you kept the default port and access it locally, you can use:

   ```
   https://localhost:4849
   ```

   The Admin Console opens in your browser.

   a. On the left, expand the *Applications* folder.

   b. On the left, click *Enterprise Applications*.

   c. On the right, click *Deploy*.

   d. Specify the *iwjca15cci.ear* file, for example:

   ```
   C:\Program Files\iWay55\etc\setup\iwjca15cci.ear
   ```

3. Click *Next*.
4. Ensure the Application Name is:

   ```
   iwjca15cci
   ```

5. Click *OK*.

   The sample JCA 1.5 outbound application deploys.

   The application is expanded into a directory under your domain, for example:

   ```
   C:\Sun\AppServer\domains\domain1\applications\j2ee-apps
   \iwafjca15-samples
   ```

Procedure: How to Use the Outbound Sample JCA 1.5 Application

The outbound sample application is a Web servlet that can be accessed through a browser.

To access the outbound sample application:

1. Ensure that your application server is running.
2. Enter the following URL in your browser:

   ```
   http://hostname:port/iwjca15cci/index.jspiwae/index.html
   ```

   where:

   ```
   hostname
   ```

   Is the name of the machine where your application server is running.

`port`

Is the port for the domain you are using

3. Provide the following information:

   Adapter. The name of the adapter you want to test.

   Target. The name of a target you have already configured for that adapter.

   User name. A user name to access the back end system.

   InputDoc. Insert a sample input document

4. Choose the appropriate Transaction option.

   The output appears in your browser.

APPENDIX A

Servlet Sample Code

Topic:

- iWay Servlet Sample Code

This section contains iWay servlet sample code for the iWay Connector for J2EE Connector Architecture (iWay Connector for JCA) version 1.0.

iWay Servlet Sample Code

The following are examples of iWay servlet code for the iWay Connector for JCA version 1.0.

Example: Error.jsp

The following is an example of Error.jsp code.

```
<%@ page contentType="text/html" isErrorPage="true" %>


<%@ page import="java.util.*, java.io.*" %>
<%@ page import="com.ibi.adapters.*" %>
<%@ page import="javax.resource.*" %>

<%!
  // CallStack to String
  public static String getStackTrace(Throwable t) {

     if(t == null) {
        return "Callstack not available.";
     }

      String stackTrace = null;

      try {
        StringWriter sw = new StringWriter();
        PrintWriter pw = new PrintWriter(sw);
        t.printStackTrace(pw);
        pw.close();
        sw.close();
        stackTrace = sw.getBuffer().toString();
      }
      catch(Exception ex) {}
         return stackTrace;
     }
%>

<!-- ========================================================== -->
<!--                 HEADER                                     -->
<!-- ========================================================== -->

<html>
   <head>
      <title>IWAF JCA Sample </title>
      <link href="default.css" rel="stylesheet" media="screen">
   </head>

   <body leftmargin="0" topmargin="0" marginwidth="0" marginheight="0"
```

```
text="#000000">

   <%@ include file="header.html" %>


   <TABLE WIDTH="100%" BORDER="0" CELLSPACING="0" CELLPADDING="0">
     <TR>

       <%-- ------------------Begin Side ------------ --%>
       <TD WIDTH="130" VALIGN="top" ALIGN="RIGHT"/>
       <TD WIDTH="5%"> </TD>

       <%-- ------------------Begin Center ------------ --%>
       <TD WIDTH="85%" VALIGN="top">
         <br>
         <H1 CLASS="orange">ERROR PAGE</H1>

         <span class="body">
            <%
               String message =
(String)request.getAttribute("iway.message");
               if (message != null) {
                  out.print(message);
               } else {
            %>
              Notify your administrator with the information below.
              <% } %>
         </span>

         <h2><%= new Date() %></h2>

         <%
         Exception ae = null;  // AdapterException
         Throwable vt = null;  // VendorException
         if (exception != null) {
            if (exception instanceof ResourceException) {
               ae = ((ResourceException)exception).getLinkedException();
               if (ae instanceof AdapterException) {
                  vt = ((AdapterException)ae).getVendorThrowable();
               }
            }
            if (ae != null) {
               out.println("<h2> AdapterException: " + ae.getMessage() +
"</h2>");
               out.println("<p>" + getStackTrace(ae) + "</p>");
               if (vt != null) {
                     out.println("<h2> VendorException: " +
vt.getMessage() + "</h2>");
```

```
                        out.println("<p>" + getStackTrace(vt) + "</p>");
                    }
                }
            %>
            <h2>IWAFJCAException: <%= exception.getMessage() %></h2>
            <p><%= getStackTrace(exception) %></p>
            <% } %>

          </td>
        </tr>
      </table>

<!-- ========================================================== -->
<!--                 FOOTER                                     -->
<!-- ========================================================== -->

   <%@ include file="footer.html" %>

      </body>
</html>
```

Example: Event.jsp

The following is an example of Event.jsp code.

```
  <%@ page error
Page="error.jsp" %>
<%@ page contentType="text/html" %>

<%@ include file="includes/include_util.jsp" %>
<%@ include file="includes/include_jca.jsp" %>
<%@ include file="includes/include_ae.jsp" %>

<%

////////////////////////////////////////////////////////////////////////
ConnectionFactory cf = (ConnectionFactory)application.getAttribute("cf");
if (cf == null) {
   log(application, "service.jsp: Obtaining IWAF JCA Connection
factory,");
   cf = getConnectionFactory(application);
}
////////////////////////////////////////////////////////////////////////

// Only do it once
String[] adapterNames = (String[])session.getAttribute("e.adapterNames");
if (adapterNames == null) {
   IDocument doc = aeCall(cf, "<GETADAPTERINFO/>");
   adapterNames = getEventAdapterNames(doc);
   session.setAttribute("e.adapterNames", adapterNames);
}

// Parsing request parameters
// Parsing request parameters
String adapter = request.getParameter("adapter");
if (adapter == null && adapterNames.length > 0) {
   adapter = adapterNames[0];
}

// Only do it if it is not a session
String[] channelNames =
(String[])session.getAttribute("event.channelNames");
String sessionAdapter = (String)session.getAttribute("event.adapter");
if (channelNames == null || (!adapter.equals(sessionAdapter))) {
   IDocument doc = aeCall(cf, "<GETCHANNELINFO><target>" + adapter + "</
target></GETCHANNELINFO>");
   channelNames = getChannelNames(doc);
   session.setAttribute("event.adapter", adapter);
   session.setAttribute("event.channelNamesNames", channelNames);
}
```

```
String channel = request.getParameter("channel");
if (channel == null && channelNames.length > 0) {
   channel = channelNames[0];
}

String action  = request.getParameter("action");
if (channel != null && action != null) {
   if (action.equals("start")) {
      aeCall(cf, "<STARTCHANNEL><target>" + adapter + "</target><name>" +
channel + "</name></STARTCHANNEL>");
   } else {
      aeCall(cf, "<STOPCHANNEL><target>" + adapter + "</target><name>" +
channel + "</name></STOPCHANNEL>");
   }
}

IDocument channelDoc = null;
if (channel != null) {
   channelDoc = aeCall(cf, "<GETCHANNELSTATUS><target>" + adapter + "</
target><name>" + channel + "</name></GETCHANNELSTATUS>");
}


%>

<!-- ========================================================== -->
<!--                 HEADER                                     -->
<!-- ========================================================== -->

<html>
   <head>
      <title>IWAF JCA Sample </title>
      <link href="default.css" rel="stylesheet" media="screen">
   </head>

   <body leftmargin="0" topmargin="0" marginwidth="0" marginheight="0"
text="#000000">

   <%@ include file="header.html" %>

<!-- ========================================================== -->
<!--                 BODY                                       -->
<!-- ========================================================== -->

      <TABLE WIDTH="100%" BORDER="0" CELLSPACING="0" CELLPADDING="0">
        <TR>
            <TD WIDTH="5%" VALIGN="top" ALIGN="RIGHT" />
```

```
            <TD WIDTH="30%" VALIGN="top">

               <!-- Adapters -->
               <br/>
               <h4>Event Adapters</h4>
               <p>Click the adapter below to see available channels
(configurations).</p>
               <ul>
                  <%
                     for (int i = 0; i < adapterNames.length; i++) {
                  %>
                   <li>
                    <a href="event.jsp?adapter=<%= adapterNames[i] %>"
class="top_menu"><%= adapterNames[i] %></a>
                   </li>
                  <%
                   }
                   // When adapter not provided, pick the first one.
                   if (adapter == null && adapterNames.length > 0) {
                       adapter = adapterNames[0];
                   }
                  %>
               </ul>

               <!-- Channels for adapter -->
               <br/>
               <h4>Channels for <b><%= adapter %></b></h4>
               <%
                  if (channelNames.length <= 0) {
                     out.print("<p>No Channels configured for this
adapter.</p>");
                  } else {
               %>
               <ul>
                  <% for (int i = 0; i < channelNames.length; i++) { %>
                   <li>
                    <a href="event.jsp?adapter=<%= adapter %>&channel=<%=
channelNames[i] %>"  class="top_menu"><%= channelNames[i] %></a>
                     <a href="event.jsp?adapter=<%= adapter %>&channel=<%=
channelNames[i] %>&action=start"  class="top_menu"> (start)</a>
                      <a href="event.jsp?adapter=<%= adapter %>&channel=<%=
channelNames[i] %>&action=stop"  class="top_menu"> (stop)</a>
                    </li>
                  <% } /* for */
                  %>
               </ul>
               <% } /* else */   %>
```

```
            </TD>

            <% if (channel != null) { %>
               <TD WIDTH="65%" VALIGN="top">
                  <br/>
                  <h4>Adapter Status for <%= adapter+" channel "+channel
%></h4>

                  <% if (channelDoc != null) { %>
                     <h4>Status</h4>
                     <table>
                     <%
                        INode statusNode =
channelDoc.getRootTree().getFirstChildNode();
                        String isactive =
statusNode.findChildByName("isactive").getValue();
                        out.println("<tr><td>Active:</td><td>");
                        out.println(isactive);
                        out.println("</td></tr><tr><td>Init. time:</
td><td>");

out.println(long2Date(statusNode.findChildByName("inittime").getValue()))
;

                        if (isactive.equalsIgnoreCase("true")) {
                           out.println("</td></tr><tr><td>Activate
time:</td><td>");
                           String actTime =
statusNode.findChildByName("activatetime").getValue();
                           out.println(long2Date(actTime));
                           out.println("</td></tr><tr><td>Elapsed time:</
td><td>");
                           out.println(elapseString(actTime));
                           out.println("</td></tr><tr><td>Service
count:</td><td>");

out.println(statusNode.findChildByName("servicecount").getValue());
                           out.println("</td></tr><tr><td>Error count:</
td><td>");

out.println(statusNode.findChildByName("errorcount").getValue());
                           out.println("</td></tr><tr><td>Event count:</
td><td>");

out.println(statusNode.findChildByName("eventcount").getValue());
                           out.println("</td></tr><tr><td>Avg. service
time (msec):</td><td>");
```

```
out.println(statusNode.findChildByName("avgservicetime").getValue());
                           out.println("</td></tr><tr><td>Last service
time (msec):</td><td>");

out.println(statusNode.findChildByName("lastservicetime").getValue());
                         }
                      %>
                   <% } %>
                   </td></tr></table>
                   <br/>
                </TD>
             <% } %>

          </TR>
       </table>

<!-- ========================================================== -->
<!--                 FOOTER                                     -->
<!-- ========================================================== -->

   <%@ include file="footer.html" %>

   </body>
</html>
```

Example: Include_ae.jsp

The following is an example of Include_ae.jsp code.

```
<%@ page import="java.u   til.*" %>


<%!

/*
 * Static helper methods for parsing IWAE messages
 */

// Based on GETADAPTERINFOResponse document
// @return array of available service adapters
private static String[] getAdapterNames(IDocument doc)
{

   List adapterNamesList = new ArrayList();

  INode rootNode       = doc.getRootTree();
  INode adapterNode = rootNode.findNodeByPath("/GETADAPTERINFOResponse/
adapter");

  while(adapterNode != null) {

      // Getting adapters
      adapterNode.snipNode(); // Not to mess up next find.
      INode targetNode = adapterNode.findChildByName("target");

      // Add the adapters with design descriptor to the list
      List descriptors = adapterNode.getAllChildren("descriptor");
      for (Iterator i = descriptors.iterator(); i.hasNext(); ) {
         INode descriptorNode = (INode)i.next();
         String attrFormat = descriptorNode.getAttribute("format");
         if (attrFormat != null && attrFormat.equals("design")) {
            adapterNamesList.add(targetNode.getValue());
            break;
         }
      } // for

      // Finding next adapter
      adapterNode = rootNode.findNodeByPath("/GETADAPTERINFOResponse/
adapter");
  }

  String[] names = new String[adapterNamesList.size()];
  names = (String[])adapterNamesList.toArray(names);
```

```
  return names;
}

// Based on GETADAPTERINFOResponse document
// @return array of available service adapters
private static String[] getEventAdapterNames(IDocument doc)
{

   List adapterNamesList = new ArrayList();

  INode rootNode       = doc.getRootTree();
  INode adapterNode = rootNode.findNodeByPath("/GETADAPTERINFOResponse/
adapter");

  while(adapterNode != null) {

      // Getting adapters
      adapterNode.snipNode(); // Not to mess up next find.
      INode targetNode = adapterNode.findChildByName("target");

      // Add the adapters with design descriptor to the list
      List descriptors = adapterNode.getAllChildren("descriptor");
      for (Iterator i = descriptors.iterator(); i.hasNext(); ) {
         INode descriptorNode = (INode)i.next();
         String attrFormat = descriptorNode.getAttribute("format");
         if (attrFormat != null && attrFormat.equals("event")) {
            adapterNamesList.add(targetNode.getValue());
            break;
         }
      } // for

      // Finding next adapter
      adapterNode = rootNode.findNodeByPath("/GETADAPTERINFOResponse/
adapter");
  }

  String[] names = new String[adapterNamesList.size()];
  names = (String[])adapterNamesList.toArray(names);

  return names;
}

// Based on GETTARGETINFOResponse
// @return array of available configurations for adapter
private static String[] getTargetNames(IDocument doc)
{

   List targetNamesList = new ArrayList();
```

```
  INode rootNode   = doc.getRootTree();
  INode targetNode = rootNode.findNodeByPath("/GETTARGETINFOResponse/
target");

  while(targetNode != null) {

      // Getting adapters
      targetNode.snipNode(); // Not to mess up next find.
      INode nameNode = targetNode.findChildByName("name");
      targetNamesList.add(nameNode.getValue());

      // Finding next adapter
      targetNode = rootNode.findNodeByPath("/GETTARGETINFOResponse/
target");
  }

  String[] names = new String[targetNamesList.size()];
  names = (String[])targetNamesList.toArray(names);

  return names;
}

// Based on GETTARGETINFOResponse
// @return array of available configurations for adapter
private static String[] getChannelNames(IDocument doc)
{

   List targetNamesList = new ArrayList();

  INode rootNode   = doc.getRootTree();
  INode targetNode = rootNode.findNodeByPath("/GETCHANNELINFOResponse/
channel");

  while(targetNode != null) {

      // Getting adapters
      targetNode.snipNode(); // Not to mess up next find.
      INode nameNode = targetNode.findChildByName("name");
      targetNamesList.add(nameNode.getValue());

      // Finding next adapter
      targetNode = rootNode.findNodeByPath("/GETCHANNELINFOResponse/
channel");
  }

  String[] names = new String[targetNamesList.size()];
  names = (String[])targetNamesList.toArray(names);
```

```
  return names;
}

%>
```

Example: Include_jca.jsp

The following is an example of Include_jca.jsp code.

```
<%@ page import="com.ibi.common.*, com.ibi.afjca.cci.*,
com.ibi.afjca.spi.*" %>
<%@ page import="javax.resource.*,
javax.resource.cci.*,javax.resource.spi.*" %>
<%@ page import="javax.naming.*" %>

<%!

////////////////////////////////////////////////////////////////////////
/
//JCA
////////////////////////////////////////////////////////////////////////
/

// Obtain ConnectionFactory from JNDI
private ConnectionFactory getConnectionFactory(ServletContext
application)
   throws ResourceException, NamingException
{
      ConnectionFactory cf = null;

   String iwayHome     = application.getInitParameter("iway.home");
   String iwayConfig   = application.getInitParameter("iway.config");
   String iwayLoglevel = application.getInitParameter("iway.loglevel");

   String iwayJndi     = application.getInitParameter("iway.jndi");

   application.log("iway.home    : " + iwayHome);
   application.log("iway.config  : " + iwayConfig);
   application.log("iway.loglevel: " + iwayLoglevel);
   application.log("iway.jndi    : " + iwayJndi);

   if (iwayJndi != null && iwayJndi.trim().length() > 0) {
      InitialContext context = new InitialContext();
      cf = (ConnectionFactory)context.lookup(iwayJndi);
      application.setAttribute("non-managed", Boolean.FALSE);
   } else {
      IWAFManagedConnectionFactory mcf = new
IWAFManagedConnectionFactory();
      mcf.setIWayHome(iwayHome);
      mcf.setIWayConfig(iwayConfig);
      mcf.setLogLevel(iwayLoglevel);
      cf = (ConnectionFactory)mcf.createConnectionFactory();
      application.setAttribute("non-managed", Boolean.TRUE);
   }
```

```
   application.setAttribute("cf", cf);
   return cf;

}

// Obtain Connection for design time
private Connection getDesigntimeConnection(ConnectionFactory cf)
   throws ResourceException
{
   // Create connectionSpec
   IWAFConnectionSpec cs = new IWAFConnectionSpec();
   cs.setAdapterName("IAEAdapter"); // Special Adapter

   return cf.getConnection(cs);
}

// Process design time message
private IWAFRecord executeDesignInteraction(Connection c, Record r)
   throws ResourceException
{
   // Create interaction
   Interaction i = c.createInteraction();

   // Create interactionSpec for DESIGNTIME
   IWAFInteractionSpec is = new IWAFInteractionSpec();
   is.setFunctionName("IWAE");  // AE Function, only available for
IAEAdapter

   // Execute
   return (IWAFRecord)i.execute(is, r);
}

////////////////////////////////////////////////////////////////////////
/

private static Connection getRuntimeConnection(ConnectionFactory cf,
  String adapterName, String configuration)
   throws ResourceException
{
   // Create connectionSpec
     // Aidong - Maybe we need a designtime connection spec.
   IWAFConnectionSpec cs = new IWAFConnectionSpec();
   cs.setAdapterName(adapterName);
   cs.setConfig(configuration);

   return cf.getConnection(cs);
}
```

```
// Execute Service Call
private static Record executeRunInteraction(Connection c, Record r)
   throws ResourceException
{
   // Create interaction
   Interaction i = c.createInteraction();

   // Create interactionSpec for RUNTIME
   IWAFInteractionSpec is = new IWAFInteractionSpec();
   is.setFunctionName("PROCESS");

   // Execute
   return i.execute(is, r);
}

////////////////////////////////////////////////////////////////////////
//

// Execute AE Call
private IDocument aeCall(ConnectionFactory cf, String message) throws
Exception {

   //log("Processing design time message: " + message);
   Connection c = getDesigntimeConnection(cf);

   IWAFRecord rIn = new IWAFRecord("input");
   rIn.setRootXML(message);
   IWAFRecord resRec = executeDesignInteraction(c, rIn);

   c.close();
   return resRec.getIDocument();
}


// Execute Service Call
private String serviceCall(ConnectionFactory cf, String message, String
adapter, String config)
   throws Exception
{
   //log("Processing service message: " + message);
   Connection c = getRuntimeConnection(cf, adapter, config);

   /*
   IWAFRecord rIn = new IWAFRecord("input");
   rIn.setRootXML(message);
   IWAFRecord docRec = executeRunInteraction(c, rIn);
```

```
   */

   IndexedRecord rIn =
cf.getRecordFactory().createIndexedRecord("input");
   rIn.add(message);
   IndexedRecord response = (IndexedRecord) executeRunInteraction(c,
rIn);

/*
   // Using IWAFRecord to get IDocument. Aidong is fixing this limitation
   IWAFRecord docRec = new IWAFRecord();
   docRec.setRootXML((String)response.get(0));
*/
   c.close();
   return (String)response.get(0);
}

%>
```

Example: Include_util.jsp

The following is an example of Include_util.jsp code.

```
<%@ page import="java.io
.*, java.util.*" %>

<%!

// CallStack to String
public static String getStackTrace(Throwable t) {

  String stackTrace = null;

  try {
      StringWriter sw = new StringWriter();
   PrintWriter pw = new PrintWriter(sw);
   t.printStackTrace(pw);
   pw.close();
   sw.close();
   stackTrace = sw.getBuffer().toString();
  }
  catch(Exception ex) {}
  return stackTrace;
}

// Web Server logging
public static void log(ServletContext ctx, String msg) {
   ctx.log(msg);
}

// long string to date string
private static String long2Date(String millis) {
   Date date = new Date(Long.parseLong(millis));
   return date.toString();
}

// TODO: It got to be a better way of doing it.
private static String elapseString(String aMillis) {

   StringBuffer sb = new StringBuffer();

   // In seconds
   long runningTime = System.currentTimeMillis() -
Long.parseLong(aMillis);
   runningTime = runningTime / 1000;  // seconds

   long days, hours, minutes, seconds;
```

```
   days = runningTime / (86400);
   if (days > 0) {
      sb.append(days);
      sb.append(" days ");
      runningTime = runningTime - (days * 86400);
   }

   hours = runningTime / (3600);
   if (hours > 0) {
      sb.append(hours);
      sb.append(" hours ");
      runningTime = runningTime - (hours * 3600);
   }

   minutes = runningTime / 60;
   if (minutes > 0) {
      sb.append(minutes);
      sb.append(" min(s) and ");
      runningTime = runningTime - (minutes * 60);
   }

   seconds = runningTime;
   sb.append(seconds);
   sb.append(" sec(s)");

   return sb.toString();
}

%>
```

Example: Index.jsp

The following is an example of Index.jsp code.

```
<%@ page content
Type="text/html" %>

<!-- ========================================================== -->
<!--                 HEADER                                     -->
<!-- ========================================================== -->

<html>
   <head>
         <title>IWAF JCA Sample </title>
      <link href="default.css" rel="stylesheet" media="screen">
   </head>

   <body leftmargin="0" topmargin="0" marginwidth="0" marginheight="0"
text="#000000">

   <%@ include file="header.html" %>

<!-- ========================================================== -->
<!--                 BODY                                       -->
<!-- ========================================================== -->

      <TABLE WIDTH="100%" BORDER="0" CELLSPACING="0" CELLPADDING="0">
        <TR>
            <TD WIDTH="130" VALIGN="top" ALIGN="RIGHT" />

            <TD WIDTH="85%" VALIGN="top">

<!--
               <br/>
               <h4>Remote management</h4>
               <p>This allows the remote management of the JCA Adapter
via
               IWAE using a SOAP listener.
               </br><b>IMPORTANT: Security is currently disabled</b>
               </p>
               <ul>
                   <li><a href="#"  class="top_menu">Configure</a></li>
                   <li><a href="#"  class="top_menu">Monitor</a></li>
                   <li><a href="#"  class="top_menu">Start/Stop</a></li>
               </ul>
-->
               <br/>
               <h4>Testing tools</b></h4>
               <ul>
```

```
                    <li><a href="service.jsp"  class="top_menu">Service
adapters</a></li>
                    <li><a href="event.jsp"  class="top_menu">Event
adapters</a></li>
                </ul>
            </TD>

        </TR>
      </table>

<!-- ========================================================== -->
<!--                 FOOTER                                     -->
<!-- ========================================================== -->

   <%@ include file="footer.html" %>

   </body>
</html>
```

Example: Service.jsp

The following is an example of Service.jsp code.

```
<%@ page errorPage= "error.jsp" %>

<%@ page contentType="text/html" %>

<%@ include file="includes/include_util.jsp" %>
<%@ include file="includes/include_jca.jsp" %>
<%@ include file="includes/include_ae.jsp" %>

<%

//////////////////////////////////////////////////////////////////////
ConnectionFactory cf = (ConnectionFactory)application.getAttribute("cf");
if (cf == null) {
      log(application, "service.jsp: Obtaining IWAF JCA Connection
factory,");
   cf = getConnectionFactory(application);
}
//////////////////////////////////////////////////////////////////////

// Only do it once
String[] adapterNames = (String[])session.getAttribute("e.adapterNames");
if (adapterNames == null) {
   IDocument doc = aeCall(cf, "<GETADAPTERINFO/>");
   adapterNames = getAdapterNames(doc);
   session.setAttribute("e.adapterNames", adapterNames);
}

// Parsing request parameters
String adapter = request.getParameter("adapter");
if (adapter == null && adapterNames.length > 0) {
   adapter = adapterNames[0];
}

String target  = request.getParameter("target");

// Only do it if it is not a session
String[] targetNames =
(String[])session.getAttribute("service.targetNames");
String sessionAdapter = (String)session.getAttribute("service.adapter");
if (targetNames == null || (!adapter.equals(sessionAdapter))) {
   IDocument doc = aeCall(cf, "<GETTARGETINFO><target>" + adapter + "</
target></GETTARGETINFO>");
targetNames = getTargetNames(doc);
      session.setAttribute("service.adapter", adapter);
   session.setAttribute("service.targetNames", targetNames);
```

```
}

////////////////////////////////////////////////////////////////////////
// Service call
////////////////////////////////////////////////////////////////////////
String input       = request.getParameter("input");
String output      = null;
long   elapsedTime = 0;

if (input != null) {
   long stime = System.currentTimeMillis();
   output = serviceCall(cf, input, adapter, target);
   elapsedTime = System.currentTimeMillis() - stime;
}

%>

<!-- ========================================================== -->
<!--                 HEADER                                     -->
<!-- ========================================================== -->

<html>
   <head>
      <title>IWAF JCA Sample </title>
      <link href="default.css" rel="stylesheet" media="screen">
   </head>

   <body leftmargin="0" topmargin="0" marginwidth="0" marginheight="0"
text="#000000">

   <%@ include file="header.html" %>

<!-- ========================================================== -->
<!--                 BODY                                       -->
<!-- ========================================================== -->

      <TABLE WIDTH="100%" BORDER="0" CELLSPACING="0" CELLPADDING="0">
        <TR>
            <TD WIDTH="5%" VALIGN="top" ALIGN="RIGHT" />

            <TD WIDTH="30%" VALIGN="top">

               <!-- Adapters -->
               <br/>
               <h4>Service Adapters</h4>
               <p>Click the adapter below to see available targets
(configurations).</p>
               <ul>
```

```
                    <%
                       for (int i = 0; i < adapterNames.length; i++) {
                    %>
                     <li>
                      <a href="service.jsp?adapter=<%= adapterNames[i] %>"
class="top_menu"><%= adapterNames[i] %></a>
                     </li>
                    <%
                     }
                    %>
                 </ul>

                 <!-- Targets for adapter -->
                 <br/>
                 <h4>Targets for <b><%= adapter %></b></h4>
                 <%
                    if (targetNames.length <= 0) {
                       out.print("<p>No targets configured for this
adapter.</p>");
                    } else {
                 %>
                 <ul>
                    <% for (int i = 0; i < targetNames.length; i++) { %>
                     <li>
                      <a href="service.jsp?adapter=<%= adapter
%>&target=<%= targetNames[i] %>"  class="top_menu"><%= targetNames[i]
%></a>
                     </li>
                    <% } /* for */

                    // When target not provided, pick the first one.
                    if (target == null && targetNames.length > 0) {
                       target = targetNames[0];
                    }
                    %>
                 </ul>
                 <% } /* else */   %>
              </TD>

              <% if (target != null) { %>
                 <TD WIDTH="65%" VALIGN="top">
                    <br/>
                    <h4>Request for <%= adapter+" target "+target %></h4>
                    <FORM ACTION="service.jsp" METHOD="POST">
                       <input type='hidden' name='adapter' value='<%=
adapter %>' />
                      <input type='hidden' name='target' value='<%= target
%>' />
```

```
                    <br/><textarea name="input" rows="10" cols="50"><%=
(input==null?"":input) %></textarea>
                    <br/><input type="submit" value="Send"/><input
type="reset"/>
                 <% if (output != null) { %>
                    <h4>Response in <%= elapsedTime %> msecs</h4>
                    <textarea name="output" rows="10" cols="50"
disable="true"><%= output %></textarea>
                 <% } %>
              </FORM>
              <br/>
              </TD>
           <% } %>

       </TR>
     </table>

<!-- ========================================================== -->
<!--                 FOOTER                                     -->
<!-- ========================================================== -->

   <%@ include file="footer.html" %>

   </body>
</html>
```

Example: IWAFJCADemo.java

The following is an example of IWAFJCADemo.java code.

```
import javax.resource.*;
import javax.resource.cci.*;
import java.io.FileInputStream;
import java.io.IOException;

import com.ibi.afjca.cci.*;
import com.ibi.afjca.spi.*;

/**
 * This program demos how to use our iwaf jca 1.0 adpter in non-managed
 * environment.
 *
 * Author: Aidong Yan
 * Date: July, 2003
 */
public class IWAFJCADemo {

  private static String buildDir = "C:\\program files\\iway55";

  // IWAFManagedConnectionFactory and IWAFConnectionSpec properties:
  private static String adapterName       = "SAP";
  private static String iWayHome = buildDir;
  private static String iWayConfig= "base";
  private static String iWayRepoURL= null;
  private static String iWayRepoUser= null;
  private static String iWayRepoPassword= null;
  private static String config            = "test";
 // private static String user              = "EDAQA0";
 // private static String password          = "EDATEST3";
  private static String logLevel          = "DEBUG";

  // Request input
  private static String requestFile = "C:\\rfc1.xml";

  ////////////////////////////////////////////////////////////////////////
/////

  public static void main(String[] args) throws Exception {

    // Create connectionFactory
    // See private methods below.
    ConnectionFactory factory = (ConnectionFactory)
getConnectionFactory();
    if(factory == null) return;
```

```
    try {

      // Create connectionSpec
      IWAFConnectionSpec cs = new IWAFConnectionSpec();
      cs.setAdapterName(adapterName);
      cs.setConfig(config);
//      cs.setUserName(user);
//      cs.setPassword(password);
      cs.setLogLevel(logLevel);

      // Get Connection and Interaction to work with
      Connection  con    = factory.getConnection(cs);
      Interaction action = con.createInteraction();

      // Create interactionSpec
      IWAFInteractionSpec is = new IWAFInteractionSpec();
      is.setFunctionName(IWAFInteractionSpec.PROCESS);

      // Create Input Record
      IndexedRecord inRec =
factory.getRecordFactory().createIndexedRecord("input");
      inRec.add(readFile(requestFile));

      // Execute
      IndexedRecord outRec = (IndexedRecord) action.execute(is, inRec);

      // Handle the output
      String outStr = (String) outRec.get(0);
      System.out.println("Response:" + outStr);

      // Close connection
      con.close();

      // Shut down IWAF container only when in non-managed environment.
      ((IWAFConnectionFactory)factory).destroy();

    } catch (ResourceException re) {
      System.out.println("ResourceException:"+re);
    }
  }

  ////////////////////////////////////////////////////////////////////////
/////

  /**
   * Create one IWAFManagedConnectionFactory and config it
   */
  private static Object getConnectionFactory() {
```

```
    try {
      IWAFManagedConnectionFactory mcf = new
IWAFManagedConnectionFactory();
      mcf.setIWayHome(iWayHome);
      mcf.setIWayConfig(iWayConfig);
      mcf.setLogLevel(logLevel);
      mcf.setIWayRepoURL(iWayRepoURL);
      mcf.setIWayRepoUser(iWayRepoUser);
      mcf.setIWayRepoPassword(iWayRepoPassword);

      return mcf.createConnectionFactory();
    } catch (ResourceException re) {
      System.out.println("Couldn't create the connection factory.");
      return null;
    }
  }

  /**
   * Helper method to read file into String. Assumes the file is in the
   * machines codepage.
   */
  private static String readFile(String requestFile) throws IOException {
    FileInputStream fileIn = new FileInputStream(requestFile);
    byte[] inBytes = new byte[fileIn.available()];

    int offset = 0, counter = 0, len = inBytes.length;
    do {
      counter = fileIn.read(inBytes, offset, len);
      offset += counter;
    } while(counter != -1 && offset < len);

    return new String(inBytes);
  }

}
```

Reader Comments

In an ongoing effort to produce effective documentation, the Documentation Services staff at Information Builders welcomes any opinion you can offer regarding this manual.

Please use this form to relay suggestions for improving this publication or to alert us to corrections. Identify specific pages where applicable. You can contact us through the following methods:

Mail: Documentation Services - Customer Support
Information Builders, Inc.
Two Penn Plaza
New York, NY 10121-2898

Fax: (212) 967-0460

E-mail: books_info@ibi.com

Web form: http://www.informationbuilders.com/bookstore/derf.html

Name:_____

Company:_____

Address:_____

Telephone:_____Date:_____

E-mail:_____

Comments:

Reader Comments

Information Builders, Two Penn Plaza, New York, NY 10121-2898 **(212) 736-4433**