IBM WebSphere InterChange Server

Access Development Guide for J2EE Connector Architecture™

Version 4.3.0

IBM WebSphere InterChange Server

Access Development Guide for J2EE Connector Architecture™

Version 4.3.0

Note!
Before using this information and the product it supports, read the information in "Notices" on page 39.

30September2004

This edition of this document applies to IBM WebSphere InterChange Server (5724-178), version 4.3.0, and to all subsequent releases and modifications until otherwise indicated in new editions.

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

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

Contents

About this document

IBM[R] WebSphere[R] InterChange Server and its associated toolset are used with IBM WebSphere Business Integration adapters to provide business process integration and connectivity among leading e-business technologies and enterprise applications.

This document describes how to program a J2EE application component to use the WebSphere Access Resource Adapter (sometimes called the Resource Adapter for the IBM WebSphere InterChange Server) to access EIS applications that are managed through the IBM WebSphere InterChange Server.

Audience

This document is for IBM customers, consultants, or resellers who integrate J2EE application components with the WebSphere business integration system by using the Resource Adapter for the IBM WebSphere InterChange Server. Before you start, you should understand all the concepts (especially collaborations) explained in the manual *Technical Introduction to IBM WebSphere InterChange Server*.

Prerequisites for this document

To implement the calls to the CCI in the application component, you should know standard programming concepts and practices as well as the Java programming language. In addition, you need to follow standards as outlined in the J2EE Connector Architecture specification 1.0.

WebSphere business integration system provides the following documents that might be useful in the development of a J2EE application component that accesses an EIS application:

- The IBM WebSphere InterChange Server uses the IBM WebSphere InterChange Server Access Interface to communicate with the InterChange Server broker. For information on this API, see the *Access Development Guide*. While this manual is not a prerequisite for IBM WebSphere InterChange Server Access for J2EE to communicate with IBM WebSphere InterChange Server, knowledge of features in the Server Access Interface can be helpful.
- A J2EE application component accesses an EIS through the IBM WebSphere InterChange Server by requesting execution of a collaboration. The *Collaboration Development Guide* describes how to create and modify a collaboration.

Related documents

The complete set of documentation available with this product describes the features and components common to all WebSphere InterChange Server installations, and includes reference material on specific components.

This document contains many references to two other documents: the *System Installation Guide for Windows* or *for UNIX* and the *System Implementation Guide*. If you choose to print this document, you may want to print these documents as well.

You can install the documentation from the following sites:

- For InterChange Server documentation:
  http://www.ibm.com/websphere/integration/wicserver/infocenter
- For collaboration documentation:
  http://www.ibm.com/websphere/integration/wbicollaborations/infocenter
- For WebSphere Business Integration Adapters documentation:
  http://www.ibm.com/websphere/integration/wbiadapters/infocenter

These sites contain simple directions for downloading, installing, and viewing the documentation.

Note: Important information about this product might be available in Technical Support Technotes and Flashes issued after this document was published. These can be found on the WebSphere Business Integration Support Web site, http://www.ibm.com/software/integration/websphere/support/. Select the component area of interest and browse the Technotes and Flashes sections.

Typographic conventions

This document uses the following conventions:

`courier font`	Indicates a literal value, such as a command name, file name, information that you type, or information that the system prints on the screen.
bold	Indicates a new term the first time that it appears.
italic, *`italic`*	Indicates a variable name or a cross-reference.
blue text	Blue text, which is visible only when you view the manual online, indicates a cross-reference hyperlink. Click any blue text to jump to the object of the reference.
{ }	In a syntax line, curly braces surround a set of options from which you must choose one and only one.
`[ ]`	In a syntax line, square brackets surround an optional parameter.
`...`	In a syntax line, ellipses indicate a repetition of the previous parameter. For example, `option[,...]` means that you can enter multiple, comma-separated options.
`< >`	In a naming convention, angle brackets surround individual elements of a name to distinguish them from each other, as in `<server_name><`*`connector_name`*`>tmp.log`.
`/`, `\`	In this document, backslashes (\) are used as the convention for directory paths. For UNIX installations, substitute slashes (/) for backslashes. All WebSphere business integration system product path names are relative to the directory where the WebSphere business integration system product is installed on your system.
`%text%` and *`$text`*	Text within percent (%) signs indicates the value of the Windows `text` system variable or user variable. The equivalent notation in a UNIX environment is *`$text`*, indicating the value of the *`text`* UNIX environment variable.
ProductDir	Represents the directory where the product is installed. For IBM WebSphere InterChange Server, the default product directory is `IBM\WebSphereICS`.

New in this release

This chapter describes the following new features of this component of the WebSphere business integration system, which are covered in this document:

New in release 4.3

A new inventory and license management tool is bundled as part of WebSphere® InterChange Server. Version 2.1 of the IBM® Tivoli® License Management (ITLM) product provides a framework for this asset management. The same ITLM product is also provided with IBM WebSphere Business Integration Toolset, but is enabled only for inventory support.

New in WebSphere InterChange Server version 4.2.2

February 2004

For this release of InterChange Server, the following changes have been made to this guide:

- Chapters 1 and 2, from the previous version of this guide, have been combined into a single chapter, which provides a brief overview of the J2EE architecture as well as an introduction to resource adapters and WebSphere Server Access for J2EE. For more information, see Chapter 1, "Introduction to J2EE Connector Architecture," on page 1.
- Additional information has been provided on the `OAport` configuration property, including how to set it within the InterChange Server configuration file. For more information, see "Generating a persistent .ior file" on page 18.

December 2003

For this release of InterChange Server, the following changes have been made to this guide:

- Terminology, product names, file names, path names, and copyright information were updated in this manual for the WebSphere InterChange Server version 4.2.2 release.
- WebSphere Server Access for J2EE now uses the IBM Java Object Request Broker (ORB) to handle communication between the WebSphere Access Resource Adapter and InterChange Server. For more information, see "Handling the .ior file" on page 18.

New in WebSphere Business Integration Adapters version 2.1.x

Updated in March, 2003. The "CrossWorlds" name is no longer used to describe an entire system or to modify the names of components or tools, which are otherwise mostly the same as before. For example "CrossWorlds System Manager" is now "System Manager," and "CrossWorlds InterChange Server" is now "WebSphere InterChange Server."

The changes made in WebSphere Business Integration Adapters version 2.1 do not affect the content of this document.

New in WebSphere Business Integration Adapters version 2.x

The changes made in WebSphere Business Integration Adapters version 2.0 do not affect the content of this document.

New in release 4.1.x

The changes made in IBM CrossWorlds 4.1.0 do not affect the content of this document.

Chapter 1. Introduction to J2EE Connector Architecture

This chapter provides an introduction to the support that IBM WebSphere products provide for the Java 2 Enterprise Edition (J2EE) Connector Architecture (JCA).

Note: This section provides a high-level overview of the J2EE Connector Architecture specification 1.0.. For a more complete discussion, see the Sun Java web site at *http://java.sun.com/j2ee/connector*.

This chapter contains the following sections:

Overview of J2EE architecture

J2EE provides a set of standards for developing multi-tier enterprise applications. It includes standards for containers (run-time environments) and the services these containers provide.

Note: This section provides a high-level overview of the J2EE architecture. For a more complete discussion, see the Sun Java Web site at http://java.sun.com/j2ee.

Figure 1 shows the logical relationships in the J2EE architecture, based on a diagram in the Java 2 Platform Enterprise Edition Specification, version 1.3.

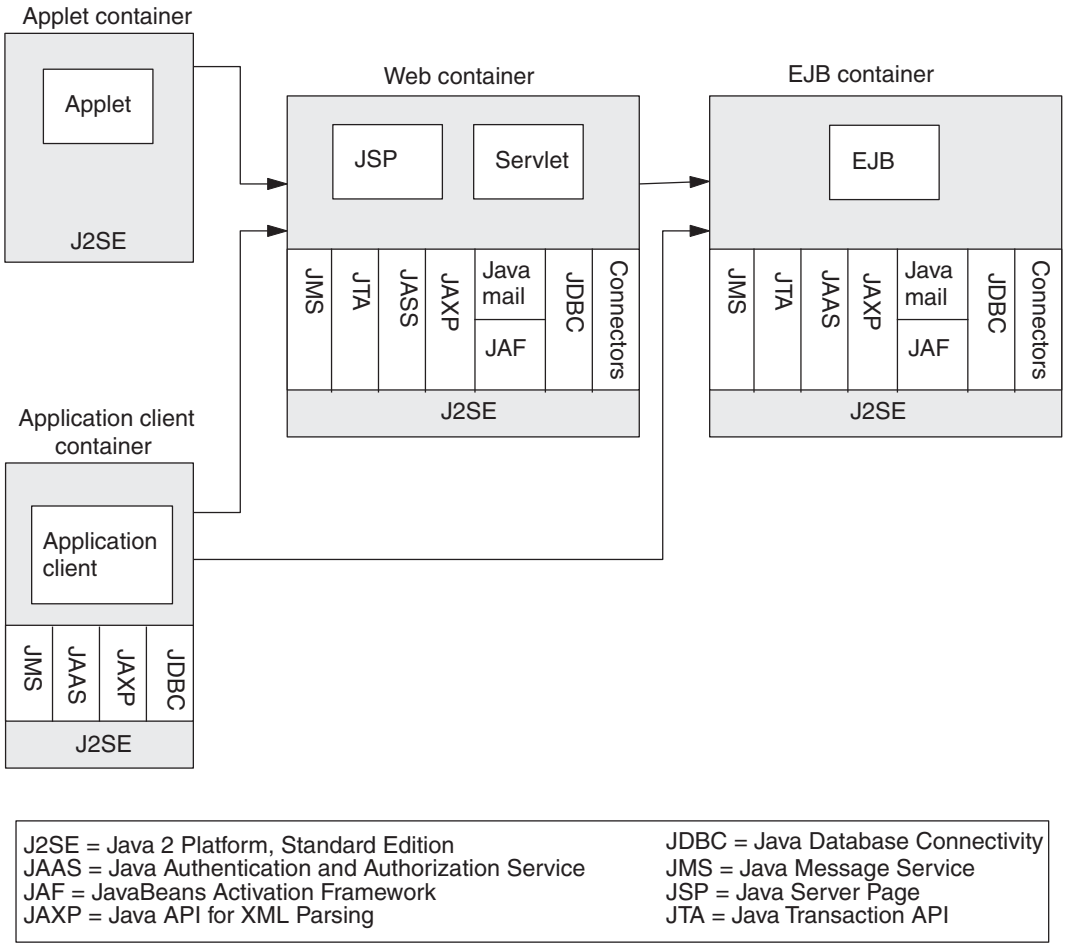

Figure 1. Logical relationships of the J2EE architecture

Overview of J2EE Connector Architecture

The J2EE Connector Architecture provides a standardized way to connect an application component with Enterprise Information Systems (EISs). Examples of an **EIS** include a legacy-database system, an ERP system, or a transaction-processing system.

Note: In the J2EE Connector Architecture specification 1.0., a *connector* is a client component that needs access to information in an EIS. This J2EE definition of *connector* differs from the WebSphere business integration system definition. In the WebSphere business integration system architecture, a *connector* is a WebSphere business integration system component that mediates between an EIS application and one or more collaborations within InterChange Server (ICS). For purposes of consistency with the J2EE Connector Architecture specification 1.0., this manual uses the J2EE definition of *connector*.

To provide standardized communication between the application component and the EIS, the J2EE Connector Architecture defines the following:

- A resource adapter
- Two environments in which you can call a resource adapter

The following sections describe each of these standards in more detail.

Resource adapter

The **resource adapter** enables application components to communicate with an EIS. The application component sends requests for the EIS to the resource adapter. The resource adapter communicates directly with the EIS, returning any requested information to the application component. The application component and resource adapter communicate with each other through the Common Client Interface (CCI). CCI is a standard client API that the J2EE Connector Architecture specification 1.0. defines. Through CCI, an application component can request the following tasks from the resource adapter:

- Establish a connection
- Request information from the EIS
- Obtain metadata
- Handle exceptions

As both Figure 2 on page 4 and Figure 3 on page 5 show, the application component and the resource adapter use CCI calls to communicate.

The resource adapter communicates directly with the EIS through the EIS-specific API. It implements the CCI interfaces through calls to this EIS-specific API. By isolating the EIS-specific API calls to the resource adapter, application components do not need to know this API. Instead, the application component uses the high-level CCI calls to request EIS services through the resource adapter.

Environments for calling a resource adapter

Table 1 shows the two environments that the J2EE Connector Architecture specification 1.0. defines, in which an application component can access an EIS through a resource adapter.

Table 1. Environments for using a resource adapter to access an EIS

J2EE connector adapter environment	Application component	Description
Managed environment	A server-side application component	Contacts an application server, which communicates with the EIS through the resource adapter
Non-managed environment	A client application	Contacts the resource adapter directly to communicate with the EIS

The following sections describe each of these environments in more detail.

Managed environment

In a managed environment, an **application server** provides basic services to J2EE application components that need to access Enterprise Information Systems (EISs). The **application component** is a server-side module that the application server manages. Examples of managed application components include an Enterprise JavaBean (EJB), a Java Server Page (JSP), or a servlet.

To communicate with an EIS, the client application notifies the appropriate application component in the application server that it needs information. The application server and the application component work together with the resource adapter to establish and manage this connection. The application server is responsible for requesting a connection to the EIS. If a connection is not available,

the application server requests one from the resource adapter associated with the EIS. The application component receives a handle for this connection.

The application server can provide interactions with many EISs because the J2EE Connector Architecture specification 1.0. defines **system-level contracts**, which define the interaction between the application server and the resource adapter. These system-level contracts define the following interactions:

- Connection management
- Security
- Transaction management

The benefit of accessing an EIS through an application server is that the application component is relieved of the need to know the details of connection management, security management, and transaction management. The application server performs these tasks and enforces them through system contracts, to which both the application server and each resource adapter must adhere.

Figure 2 shows the managed environment of the J2EE Connector Architecture, which uses an application server to manage communication between application components and resource adapters.

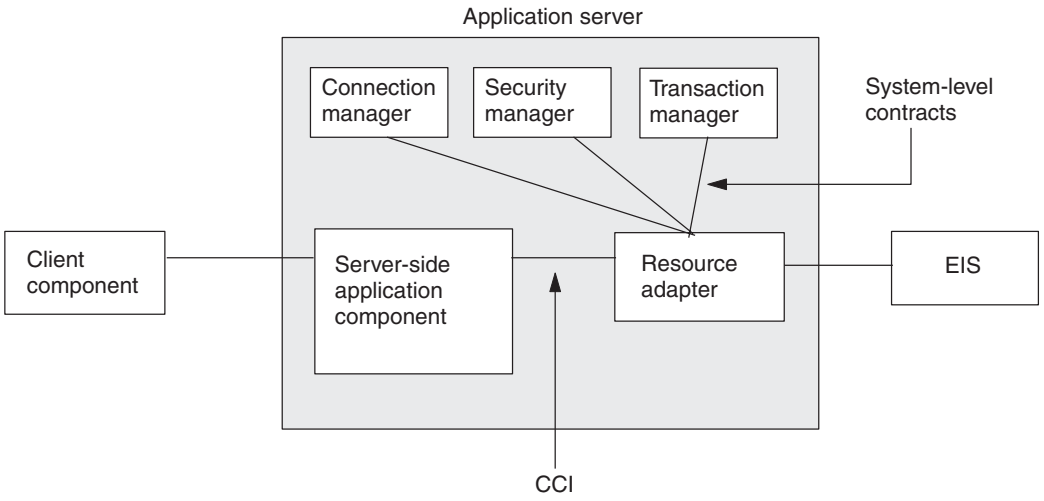

Figure 2. Managed environment: Accessing an EIS through an application server

In Figure 2, notice that the system-level contracts define the interactions between the resource adapter and the application server while the CCI defines the interaction between the resource adapter and the application component.

Non-managed environment

In a non-managed environment, the **application component** is the client application, which communicates directly with the resource adapter to access Enterprise Information Systems (EISs). Examples of non-managed application components include Java applications and applets. Figure 3 shows the non-managed environment of the J2EE Connector Architecture, in which the client application uses CCI calls to communicate with the resource adapter.

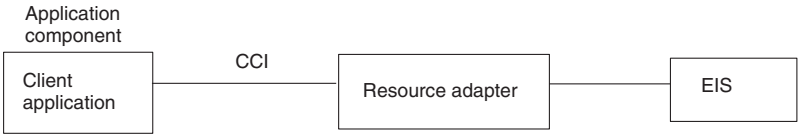

Figure 3. Non-managed environment: accessing an EIS directly

When executing in the non-managed environment, the client application must provide the connection management, security management, and transaction management.

Server Access for J2EE support for J2EE Connector Architecture

In support of the J2EE Connector Architecture specification 1.0., IBM provides WebSphere InterChange Server Access for J2EE. With this product, you instantiate a **WebSphere Access Resource Adapter** (often called just an Access Resource Adapter) within your application component. This resource adapter provides J2EE application components access to WebSphere Interchange Server (ICS). InterChange Server is the component of the WebSphere business integration system that provides the ability to maintain business processes that involve multiple EISs.

Note: For an overview of the features and capabilities of the WebSphere business integration system, see the *Technical Introduction to IBM WebSphere InterChange Server*.

The WebSphere Access Resource Adapter brings the following benefits to application components in a J2EE application:

- An application component can communicate with ICS using the same CCI calls that communicate with any other EIS whose provider implements the CCI.
- By requesting execution of WebSphere business integration system collaborations, application components can send and receive information in EIS applications that ICS manages (called ICS-managed EISs).

The following situations are possible usage scenarios for Server Access for J2EE:

- Retrieve information from an EIS application that is managed by InterChange Server.

  In an E-commerce scenario, when a customer wants to check the status on an order placed, the J2EE module (such as an EJB) responsible to fetch this information can channel the request through a WebSphere Access Resource Adapter, as follows:

  – The Access Resource Adapter initiates execution of the related collaboration.
  – Within InterChange Server, the collaboration submits the request to the respective connector, which manages the necessary retrieval from its EIS application and sends back the retrieved object to the collaboration.
  – The collaboration passes the retrieved information to the Access Resource Adapter, which then passes it up to the J2EE module that called it.

  If the same J2EE module needs to access a different EIS application (one also managed by InterChange Server) to retrieve another value, the module sends a request to the same Access Resource Adapter to execute a different collaboration. This collaboration passes on the request to the appropriate connector, which in turn manages the retrieval from its EIS application. In this way, IBM WebSphere InterChange Server Access for J2EE provides an integrated view of all the ICS-managed EIS applications.

- Support events originating from an EIS application.

  The J2EE Connector Architecture specification 1.0. does *not* support asynchronous event processing. However, you can use a combination of WebSphere business integration system components to facilitate asynchronous communication. For example, events that an EIS application originates can be used for delivering a combination of a collaboration and the WebSphere business integration system connector for JMS. The connector for JMS can deliver messages (events) to an application component as they arrive; the application component does not have to request messages in order to receive them.

Server Access for J2EE provides support for the following standards of the J2EE Connector Architecture specification 1.0.:

- Server Access for J2EE implementation of the CCI
- System-level contracts (for a managed environment only)

Note: IBM WebSphere InterChange Server Access for J2EE has been implemented in full accordance with the standards, as described in the J2EE Connector Architecture specification 1.0..

IBM WebSphere Access Resource Adapter

A WebSphere Access Resource Adapter enables an application component to access and use IBM WebSphere Business Integration Collaborations and other services. **Collaborations** represent business processes, which can involve multiple EISs. The WebSphere business integration solution initiates execution of a collaboration at the request of the Access Resource Adapter. The Access Resource Adapter sends data that represents the **triggering event** of a collaboration to the WebSphere business integration system solution, which in turn sends it to an ICS instance. Because these solutions are external to the WebSphere business integration system, their request for collaboration execution initiates a **call-triggered flow**. By requesting execution of collaborations, Access Resource Adapters can send and receive information in the ICS-managed EIS.

For an application component to communicate with a WebSphere Access Resource Adapter, Server Access for J2EE implements the CCI. To communicate with an ICS-managed EIS, an application component sends a request in the form of a CCI call to an Access Resource Adapter, which communicates with ICS to initiate the request. InterChange Server handles any communication with ICS-managed EISs through execution of the collaboration.

Figure 4 shows the WebSphere business integration system implementation of the J2EE connector architecture.

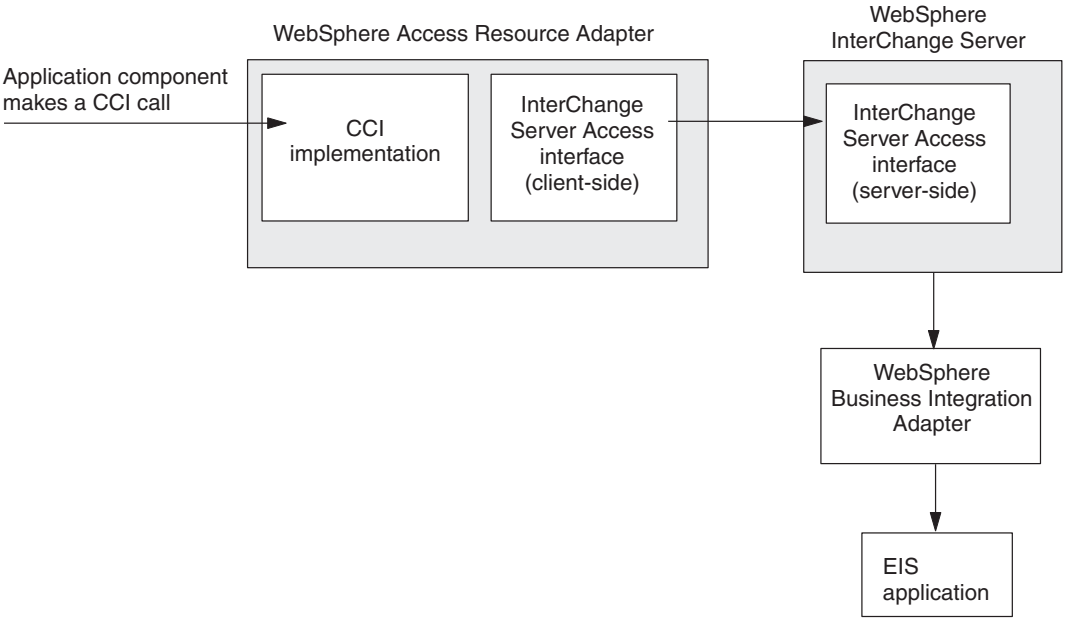

Figure 4. Access to an ICS-managed EIS through Server Access for J2EE

As Figure 4 shows, communication with an ICS-managed EIS involves an application component sending a request to an Access Resource Adapter. The Access Resource Adapter communicates with ICS to initiate the request. ICS handles any communication with ICS-managed EISs by controlling execution of the collaboration. To provide this communication, the IBM WebSphere Server Access for J2EE product includes the following components:

- Implementation of the WebSphere Access Resource Adapter

  To provide communication between the application component and the Access Resource Adapter, the application component issues calls to the CCI.
- The client-side version of WebSphere InterChange Server Access

  To provide communication between the Access Resource Adapter and ICS, each of these components uses WebSphere InterChange Server Access.

The application component and Access Resource Adapter

For an application component to communicate with a WebSphere Access Resource Adapter, Server Access for J2EE implements the CCI. To communicate with an ICS-managed EIS, an application component sends a request in the form of a CCI call to an Access Resource Adapter, which communicates with ICS to initiate the request. InterChange Server handles any communication with ICS-managed EISs through execution of the collaboration.

Table 2 provides a summary of the tasks that an application component can request from an Access Resource Adapter.

Table 2. Server Access for J2EE implementation of CCI

Task of the application component	CCI Interface	For more information
Manage a connection to WebSphere InterChange Server	`ConnectionFactoryConnectionConnectionSpec`	"Managing the connection" on page 23
Send information to an InterChange Server-managed EIS	`InteractionInteractionSpec`	"Sending data to an ICS-managed EIS" on page 27

Table 2. Server Access for J2EE implementation of CCI (continued)

Task of the application component	CCI Interface	For more information
Represent data to send to and receive from the InterChange Server-managed EIS	`Record MappedRecordIndexedRecordRecordFactory`	"Using records" on page 32
Acquire metadata	`ConnectionMetaDataResourceAdapterMetaData`	"Obtaining metadata" on page 35
Handle exceptions	`ResourceException`	"Handling exceptions" on page 37

All interfaces listed in the CCI Interface column in Table 2 are part of the following Java package:

`javax.resource.cci`

All Server Access for J2EE implementations of these CCI interfaces are part of the Java package:

`com.crossworlds.j2eeconnector`

The Access Resource Adapter and InterChange Server

To communicate with an ICS-managed EIS, a J2EE component sends a request to the WebSphere Access Resource Adapter, which communicates with ICS to initiate the request. ICS handles any communication with ICS-managed EISs by controlling execution of the collaboration. To provide communication between the Access Resource Adapter and ICS, IBM provides WebSphere InterChange Server Access.

InterChange Server Access Interface is the low-level Java interface that enables external process (called **access clients**) to request execution of a collaboration. An Access Resource Adapter is an external process to the WebSphere business integration system and therefore uses InterChange Server Access to communicate with an instance of ICS. The Server Access Interface methods use CORBA-IIOP as the underlying transport mechanism. By isolating the calls to the Server Access Interface within an Access Resource Adapter, the application component does not have to contain ICS-specific code.

Server Access for J2EE provides implementations of the CCI interfaces that use methods of the Server Access Interface. This interface enables the Access Resource Adapter to support the application-component tasks in Table 2 on page 7.. Figure 5 shows the structure of the communication between an Access Resource Adapter and ICS.

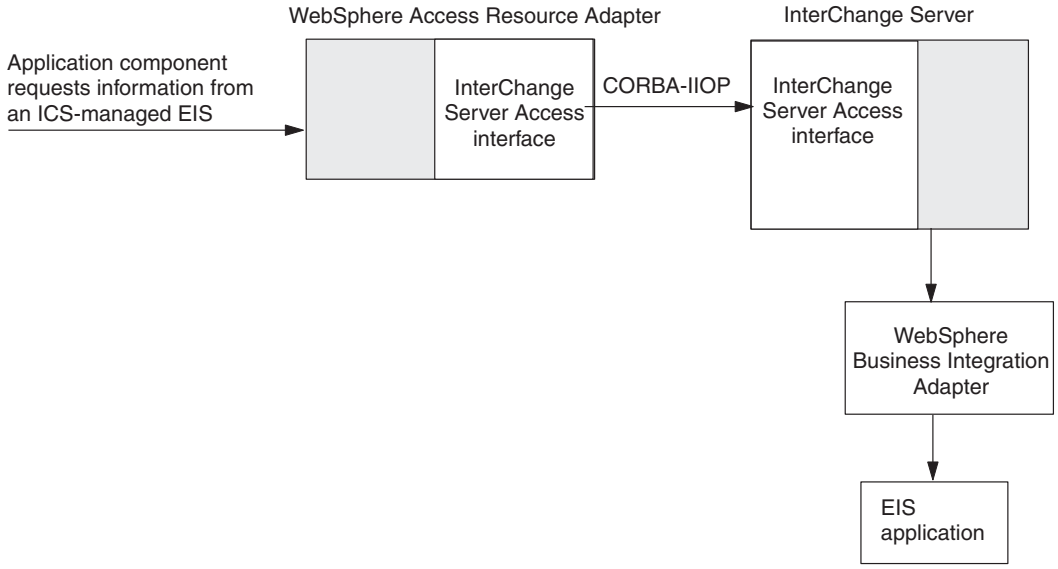

Figure 5. Access to an ICS-managed EIS

Note: For more information on the IBM WebSphere InterChange Server Access Interface, see the *Access Development Guide*.

System-level contracts

In a managed environment, the application server communicates with resource adapters to enable application components to obtain EIS information. For an application server to communicate with a WebSphere Access Resource Adapter, the resource adapter implements the system-level contracts that the J2EE Connector Architecture specification 1.0. defines. Table 3 shows the system-level contracts that IBM WebSphere InterChange Server Access for J2EE provides.

Table 3. System-level contracts supported by Server Access for J2EE

System-level contract	Server Access for J2EE support
Connection management	Yes
Security management	Minimum
Transaction management	No However, you can achieve some transaction management through the use of transactional collaborations. For more information, see the *Collaboration Development Guide*.

Overview of application-component development

This section provides the following information about the development process for a J2EE application component:

- "Stages in application-component development"
- "Development support for client components" on page 10

Stages in application-component development

To develop a J2EE application component that communicates with an ICS-managed EIS through a WebSphere Access Resource Adapter, you code the

application-component source file(s) and complete other tasks. The task of creating an application component that communicates with an ICS-managed EIS includes the following general steps:

1. Set up the development environment. Install the application-server software and IBM WebSphere InterChange Server Access for J2EE, including configuring the development environment for use with an Access Resource Adapter. For more information, see "Installing Server Access for J2EE" on page 13.
2. Within ICS, configure ports of the collaboration to be used as access and execution by call-triggered flows. This step involves configuring external collaboration ports, which handle access clients.
3. Implement and debug the application component (such as an EJB) that executes the CCI calls to use an Access Resource Adapter and execute a collaboration. Implement Java code to establish a connection, create an input record, request execution of a collaboration, and access the returned output record.

Figure 6 provides a visual overview of the application-component development process and provides a quick reference to chapters where you can find information on specific topics.

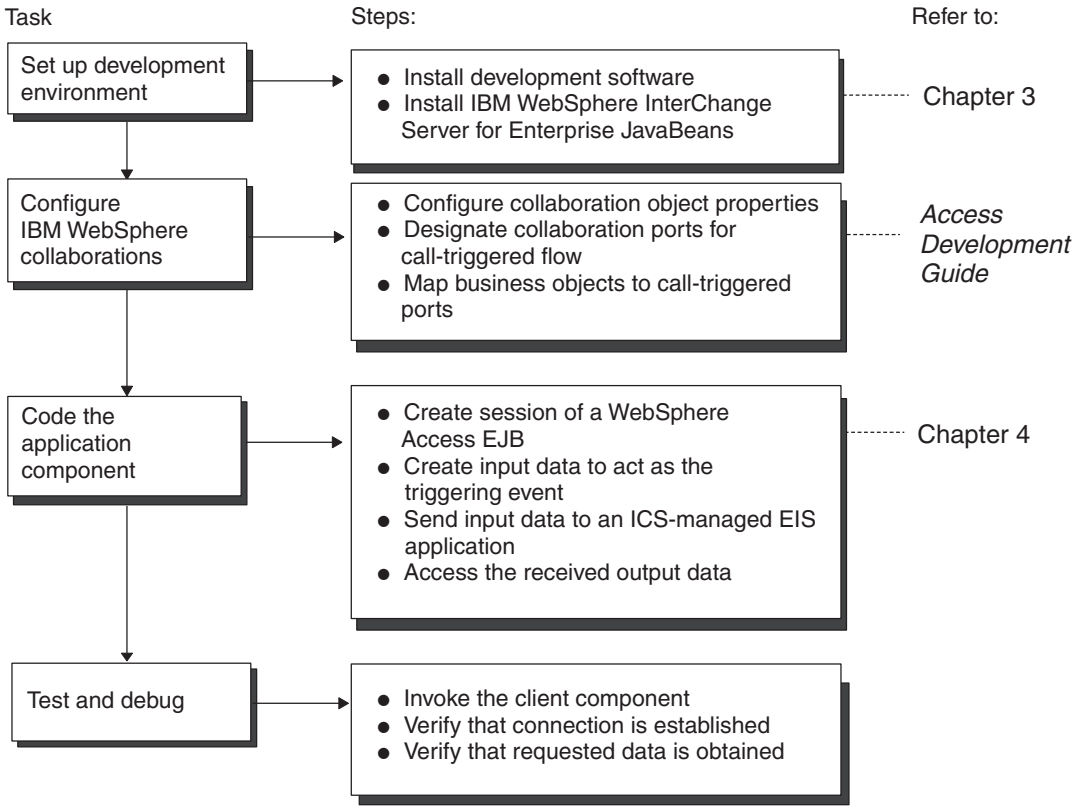

Figure 6. Overview of the application-component development task

Development support for client components

To support development of an application component that requires communication with an ICS-managed EIS, Server Access for J2EE provides the features in Table 4.

Table 4. Support for client components and an Access Resource Adapter

Server Access for J2EE support	Description
WebSphere Access Resource Adapter: `CWResourceAdapter.rar`	Provides the application component with ICS-specific implementation of CCI interfaces For more information, see "IBM WebSphere Access Resource Adapter" on page 6.
Application-component sample	Includes the following: • Sample EJB client • Sample client application that issues requests to the EJB client

To support development of a J2EE application component that uses an Access Resource Adapter, Server Access for J2EE provides the solutions in the following subdirectory of the product:

`DevelopmentKits\j2ee\ResourceAdapter\samples`

This directory contains deployable code and samples for Server Access for J2EE. The samples help you integrate a WebSphere Access Resource Adapter into the J2EE application.

Important: This sample EJB has been written to execute in the managed environment of the J2EE Connector Architecture specification 1.0..

This `samples` subdirectory contains the following files:

- The sample application component, `InventoryBean`, is an EJB client that provides the ability to look up the price of an item.
- A `readme.txt` file details how to deploy and run the EJB client.
- A Java `.class` file holds the collaboration template for the collaboration that an Access Resource Adapter initiates on behalf of the EJB client.
- A repository file contains objects that must be loaded into the InterChange Server repository.
- A sample stand-alone client application can issue requests to the EJB.
- The subdirectories listed in Table 5, contain additional sample files.

Table 5. Subdirectories included with the sample EJB client

Name	Description
src	Contains the source code for the sample EJB client and the stand-alone client application
`samples\websphere`	Contains the deployment instructions and files for WebSphere Application Server Advanced Edition 4.0

Important: For information on how to deploy and run the EJB client, refer to the `readme.txt` files included in the `samples` directory and its subdirectories.

The sample application component provides an EJB that runs within the context of the application server and can look up the adapter and use it. The EJB client demonstrates how the Access Resource Adapter can execute a WebSphere Business Integration Collaboration. This sample contains code to perform the following tasks:

- Request a connection.

- Create a record.
- Execute the interaction.
- Retrieve business data from an ICS-managed EIS.

Chapter 2. Installing and configuring Server Access for J2EE

This chapter provides information on how to install and configure IBM WebSphere InterChange Server Access for J2EE (Server Access for J2EE). It contains the following sections:

Development environment requirements

To develop an application component that uses a WebSphere Access Resource Adapter, you must have access to the following software:

- ORB Libraries
  - IBM Java Object Request Broker libraries
  - IBM Java ORB 4.5 (or compatible) service
- A Java development environment and Java 2 JDK 1.3
- Java run-time libraries 1.3
- For Server Access for J2EE in a managed environment, an application server: WebSphere Application Server, Version 4.0 Advanced Edition

  The application servers normally provide the packages for the J2EE connector development. However, if they do not include them, you can download the version 1.3 of the Java 2 SDK, Enterprise Edition (J2EE SDK) from Sun's web site.

  Note: Server Access for J2EE works with any J2EE application server that supports the J2EE Connector Architecture specification 1.0.. For this release, Server Access for J2EE has been certified with the WebSphere Application Server, as listed above.
- For Server Access for J2EE in a non-managed environment, J2EE 1.3 libraries
- The current release of InterChange Server software, release 3.1.2 or later

All of the software listed above *except* the InterChange Server software must reside on the same machine.

Note: Requirements for the run-time environment for IBM WebSphere InterChange Server Access for J2EE are the same as above *except* that the Java 2 JDK 1.3 is *not* required. However, you must have the J2EE run-time libraries 1.3.

Installing Server Access for J2EE

As part of the IBM WebSphere InterChange Server installation, InterChange Server Installer installs the files in the directories shown in Table 6. These directory paths are relative to the InterChange Server product directory.

Table 6. Installed file structure for Server Access for J2EE

Directory	Description
`DevelopmentKits\J2EE\ ResourceAdapter`	Contains the CCI implementation Server Access for J2EE is packaged into a single Resource Adapter Archive (RAR) file, called `CWResourceAdapter.rar`. This RAR file contains the following files: • `CWResourceAdapter.jar` file • Deployment descriptor: `META-INF/ra.xml` • IBM WebSphere Server Access Interface libraries
`DevelopmentKits\J2EE\ ResourceAdapter\samples`	Contains source code for the sample EJB client

Using IBM WebSphere InterChange Server Installer

To install Server Access for J2EE, run ICS Installer and select options as follows:

- To install Server Access for J2EE *on the same machine* as WebSphere InterChange Server, you can install it when you install WebSphere InterChange Server.

  Expand the Development Kits for J2EE option of Installer, and select the Resource Adapter for InterChange Server option *in addition* to other options for installing WebSphere business integration system software.
- To install Server Access for J2EE *on a different machine* from InterChange Server, run the WebSphere InterChange Server Installer on the machine on which you want to install Server Access for J2EE.

  Expand the Development Kits for J2EE option, and select *only* the Resource Adapter for InterChange Server option.

When InterChange Server Installer installs Server Access for J2EE, it copies to the machine the contents of the directories listed in Table 6. For information on InterChange Server Installer, see the *System Installation Guide for Windows* or *for UNIX*.

Deploying the WebSphere Access Resource Adapter

The following sections describe how Server Access for J2EE is deployed in each of the environments that the J2EE Connector Architecture specification 1.0. defines.

Note: For information about deploying Server Access for J2EE in a particular WebSphere business integration system-certified application server, see "Deploying the resource adapter" on page 21.

In a managed environment

In a managed environment, Server Access for J2EE can be used in either of the following ways:

- With a J2EE application that consists of one or more J2EE modules in addition to a resource-adapter module

  The J2EE Connector Architecture specification 1.0. provides requirements for the assembly and packaging of J2EE applications. The developer of the J2EE application must package and deploy it with the application.
- Directly into an application server as a stand-alone unit

  InterChange Server provides a deployment descriptor for Server Access for J2EE. This descriptor deploys the Access Resource Adapter as a stand-alone unit in the application server.

The Server Access for J2EE RAR file (CWResourceAdapter.rar) contains the deployment descriptor, ra.xml. The deployment descriptor contains a <connector> element, which is the root element of the deployment descriptor. This <connector> element contains information about the Access Resource Adapter. All general information is represented in string format.

Table 7 shows the elements in the deployment descriptor of the WebSphere Access Resource Adapter.

Table 7. Contents of the WebSphere Access Resource Adapter deployment descriptor

Deployment descriptor element	Description
<display-name>	Name of the resource adapter
<vendor-name>	Name of the vendor who provides the resource adapter
<spec-version>	Version of the J2EE Connector Architecture specification that is supported by the resource adapter
<version>	Version of the resource adapter
<resourceadapter>	Marks the beginning of the subsection within the <connector> element that provides information specific to the implementation of the resource-adapter library
<managedconnectionfactory-class>	The fully qualified name of the Java class that implements the ManagedConnectionFactory interface:javax.resource.spi.Managed-ConnectionFactory
<connectionfactory-interface>	The fully qualified name of the Java class that implements the ConnectionFactory interface:javax.resource.cci.ConnectionFactory
<connectionfactory-impl-class>	The actual implementation of the ConnectionFactory interface
<connection-interface>	The fully qualified name of the Java class that implements the Connection interface
<connection-impl-class>	The actual implementation of the Connection interface
<transaction-support>	Because IBM WebSphere InterChange Server Access for J2EE does *not* support transaction management in this release, this value is set to:NoTransaction
<config-property>	These configuration properties provide connection information for each connection request to ICS. For more information, see "Configuring Server Access for J2EE" on page 17.

Note: The ra.xml deployment descriptor is consistent with the XML DTD for the resource adapter, as specified in the J2EE Connector Architecture specification 1.0..

Figure 7 shows a sample deployment descriptor of the Access Resource Adapter, which is contained in the ra.xml file. You do not usually need to edit this file. Most application servers provide tools that provide access to the deployment descriptor.

```
<connector>
  <display-name>Resource Adapter for CrossWorlds</display-name>
  <vendor-name>CrossWorlds Software, Inc.</vendor-name>
  <spec-version>1.0</spec-version>
  <eis-type />
  <version>1.0</version>

  <resourceadapter>
    <managedconnectionfactory-class>
      com.crossworlds.j2eeconnector.CwManagedConnectionFactory
      </managedconnectionfactory-class>
    <connectionfactory-interface>
      javax.resource.cci.ConnectionFactory
      </connectionfactory-interface>
    <connectionfactory-impl-class>
      com.crossworlds.j2eeconnector.CwConnectionFactory
      </connectionfactory-impl-class>
    <connection-interface>javax.resource.cci.Connection</connection-interface>
    <connection-impl-class>
      com.crossworlds.j2eeconnector.CwConnection
      </connection-impl-class>
    <transaction-support>NoTransaction</transaction-support>
    <config-property>
      <config-property-name>IorFilename</config-property-name>
      <config-property-type>java.lang.String</config-property-type>
      <config-property-value>crossworlds.ior</config-property-value>
      </config-property>
    <config-property>
      <config-property-name>MaxConnections</config-property-name>
      <config-property-type>java.lang.Integer</config-property-type>
      <config-property-value>100</config-property-value>
      </config-property>
    <config-property>
      <config-property-name>UserName</config-property-name>
      <config-property-type>java.lang.String</config-property-type>
      <config-property-value>admin</config-property-value>
      </config-property>
    <config-property>
      <config-property-name>UserPassword</config-property-name>
      <config-property-type>java.lang.String</config-property-type>
      <config-property-value>null</config-property-value>
      </config-property>
    <authentication-mechanism>
      <authentication-mechanism-type>
        BasicPassword
        </authentication-mechanism-type>
      <credential-interface>
        javax.resource.security.PasswordCredential
        </credential-interface>
      </authentication-mechanism>
    <reauthentication-support>false</reauthentication-support>
  </resourceadapter>
</connector>
```

Figure 7. WebSphere Access Resource Adapter deployment descriptor

Important: The WebSphere Access Resource Adapter deployment descriptor in Figure 7 has been formatted differently from the actual `ra.xml` file. In this version, carriage returns have been inserted into the text to improve readability. These carriage returns do *not* appear in the actual deployment descriptor. Consult the actual `ra.xml` file on your system for the most accurate and most current version of this file.

In a non-managed environment

In a non-managed environment, the application developer must define the deployment mechanism. This developer must define a deployment environment and tool to handle deployment of all resource adapters. This deployment tool can use the deployment descriptor that InterChange Server provides in the `CWResourceAdapter.rar` file.

Configuring Server Access for J2EE

This section provides the following information for how to configure Server Access for J2EE:

- "Making Server Access for J2EE ITLM Agent enabled"
- "Setting configuration properties"
- "Handling the .ior file" on page 18
- "Configuring the application server" on page 20

Each of the steps is described in more detail in the following sections.

Making Server Access for J2EE ITLM Agent enabled

WebSphere InterChange Server Access for J2EE provides enablement for IBM Tivoli License Management (ITLM) inventory functions. In order to make the Server Access for J2EE ITLM agent enabled you must ensure that the inventory file which is installed with ICS is copied to the machine where the `CWResourceAdapter.jar` file resides. The inventory file is called

`CC001351J040300.sys`

and is found in the ProductDir/bin directory of the ICS installation.

Setting configuration properties

After the `CwManagedConnectionFactory` instance exists, invocations of the appropriate get and set methods set the connection configuration properties for the resource adapter. Table 8. lists these configuration properties.

Table 8. Server Access for J2EE configuration properties

Property name	Description	Default value
`IorFilename`	Name of the Interoperable Object Reference File (.ior) for the instance of WebSphere InterChange Server (ICS) with which the Access Resource Adapter connects For more information, see "Synchronizing the .ior file" on page 20.	`crossworlds.ior`
`MaxConnections`	Maximum number of simultaneous connections allowed to the ICS instance.	`100`
`Username`	Account name to use when connecting to the ICS instance.	`admin`
`UserPassword`	Password for the account specified in the Username property. **Note:** Rather than leaving the password as readable text in the deployment descriptor, you can specify it when requesting the connector within the application component. For more information, see "Establishing a connection" on page 24.	`null`
`ORBClassName`	Name of the Java class that implements the IBM Java Object Request Broker (ORB)	com.ibm.CORBA.iiop.ORB

Table 8. Server Access for J2EE configuration properties (continued)

Property name	Description	Default value
ORBSingletonClassName	A Java class	com.ibm.rmi.corba.ORBSingleton

The way to set these configuration properties depends on the type of environment in which Server Access for J2EE runs, as follows:

- In a managed environment—You specify configuration properties in the deployment descriptor.

  In a managed environment, the deployment descriptor specifies configuration properties. Each configuration property in the deployment descriptor provides the name, type, description, and an optional default value. The configuration properties are specified *only once* in the deployment descriptor, even though a resource adapter can be used to configure multiple connection instances.
- In a non-managed environment—The application developer must manually invoke the appropriate get and set methods to set the connection configuration properties.

  The `CwManagedConnectionFactory` interface provides `setIorFilename()` so that the client application can specify the location of the resource adapter's `.ior` file. For more information, see "In a non-managed environment" on page 37. In addition, the `ConnectionSpec` interface provides set and get methods for the user name and password (see Table 9 on page 23).

Handling the .ior file

At run time, Server Access for J2EE does not need to reside on a machine that contains InterChange Server, nor does it need to reside on the same machine as the development environment. However, for a WebSphere Access Resource Adapter to be able to locate the ICS instance it needs at run time, it must be able to locate the Object Request Broker (ORB) server, which keeps track of the locations of different CORBA objects (including InterChange Server instances) and communicates this information with ORB clients (such as a WebSphere Access Resource Adapter). To obtain the location of the ORB server, an Access Resource Adapter can use the Interoperable Object Reference File that its ICS instance generates. When ICS starts or reboots, it generates an Interoperable Object Reference file, which has the `.ior` extension. In this file, the ICS instance puts the port number on which the ORB Server listens for requests from ORB clients. The Access Resource Adapter can then use this file to locate the ORB server and, in turn, to communicate with its ICS instance.

Therefore, for an Access Resource Adapter to locate its ICS instance, you must take the following steps:

1. Request that InterChange Server generate a persistent `.ior` file.
2. Ensure that the machine on which Server Access for J2EE resides is able to locate the `.ior` file for its InterChange Server instance.
3. Synchronize the `.ior` file between the WebSphere Access Resource Adapter and InterChange Server.

Generating a persistent .ior file

When InterChange Server version 3.1.0 or later is booted up, it generates a new `.ior` file. However, by default, InterChange Server dynamically assigns a port number for the ORB server. If the port number changes each time the server boots,

the WebSphere Access Resource Adapter cannot depend on the .ior file to locate the ORB server. Therefore, an Access Resource Adapter needs ICS to generate a **persistent** .ior file.

To have InterChange Server generate a persistent ior file, you must edit the ICS configuration file (InterchangeSystem.cfg) in an XML editor and add a subsection for CORBA, if one does not already exist. Figure 8 shows the XML code that defines an *empty* CORBA subsection (one with *no* configuration parameter defined).

```
<tns:property>
   <tns:name>CORBA</tns:name>
   <tns:isEncrypted>false</tns:isEncrypted>
   <tns:updateMethod>system restart</tns:updateMethod>
   <tns:location>
      <tns:reposController>false</tns:reposController>
      <tns:reposAgent>false</tns:reposAgent>
      <tns:localConfig>true</tns:localConfig>
   </tns:location>
   XML definitions of CORBA properties go here
</tns:property>
```

Figure 8. XML definition of CORBA subsection

The CORBA subsection specifies the static port number with the OAport configuration parameter, which has the following syntax:

```
OAport=portNumber
```

For example, if the static port number is to be 15000, assign a value of 15000 to its OAport parameter in the CORBA subsection. The following XML fragment would appear within the <tns:property> tag for the CORBA subsection, in the place indicated in Figure 8 with the string "*XML definitions of CORBA properties go here*":

```
<tns:property>
  <tns:name>OAport</tns:name>
  <tns:value xml:space="preserve">15000</tns:value>
  <tns:isEncrypted>false</tns:isEncrypted>
  <tns:updateMethod>system restart</tns:updateMethod>
  <tns:location>
     <tns:reposController>false</tns:reposController>
     <tns:reposAgent>false</tns:reposAgent>
     <tns:localConfig>true</tns:localConfig>
  </tns:location>
</tns:property>
```

Important: The ICS configuration file is an XML file. To add the CORBA subsection and its configuration parameter, you must use an XML editor or must correctly format the appropriate XML tags.

For more information on the CORBA subsection in the configuration file, see the *IBM WebSphere System Installation Guide for UNIX* or *for Windows*.

Locating the .ior file

For a WebSphere Access Resource Adapter to locate the ORB server at run time, it must be able to locate the .ior file for its InterChange Server instance. Locating this file is not a problem if Server Access for J2EE and InterChange Server are on the same machine. However, if these two components are *not* on the same machine, you must take *one* of the following actions to ensure that the Access-Resource-Adapter machine can access the .ior file:

- Copy the .ior file that the InterChange Server has generated to the machine on which Server Access for J2EE (with the Access Resource Adapter) resides.
- Create a shared directory on the machine with InterChange Server and point the Access-Resource-Adapter machine to the directory.

Synchronizing the .ior file

The WebSphere Access Resource Adapter obtains the name of the InterChange Server instance with which it communicates from the ICS-specific Interoperable Object Reference (.ior) file. The resource adapter obtains the name of this .ior file from the IorFilename configuration property. Therefore, to synchronize the Access Resource Adapter and InterChange Server, you must set this IorFilename configuration property to contain the absolute path name for the ICS-specific .ior file for the machine on which Server Access for J2EE resides.

For example, to have an Access Resource Adapter connect to an InterChange Server instance named dexter, set the iorFilename environment entry to the absolute path name for this dexterinterchangeserver.ior file on the machine that contains Server Access for J2EE.

Suppose that the Server Access for J2EE deployment directory is:

ProductDir\DevelopmentKits\J2EE\ResourceAdapter

The iorFilename environment entry would contain the following string to indicate the location for the .ior file:

ProductDir\DevelopmentKits\J2EE\ResourceAdapter\dexterinterchangeServer.ior

In a managed environment, an application server can sometimes fail to recognize the absolute path name in IorFilename. If this occurs, move the .ior file to a folder within the application server's directory structure.

Configuring the application server

This section provides a summary of how to configure WebSphere Application Server (the application server that is certified for use with Server Access for J2EE) in a managed environment. For more information, refer to the following sources of information:

- The documentation for WebSphere Application Server
- The readme.txt file in the subdirectory of the Server Access for J2EE samples for the WebSphere Application Server Advanced Edition 4.0:

  ProductDir\DevelopmentKits\J2EE\ResourceAdapter\samples\websphere

The configuration of WebSphere Application Server for use with Server Access for J2EE involves the following steps:

- "Modifying configuration properties"
- "Deploying the resource adapter" on page 21
- "Making ORB libraries available" on page 21

Each of these steps is described in more detail in the following sections.

Modifying configuration properties

To update configuration properties, you change their values in the deployment descriptor for the WebSphere Access Resource Adapter, ra.xml. This deployment descriptor is part of the CWResourceAdapterWL.rar file. It is recommended that you

use the administration console of your application server once the Access Resource Adapter is deployed. You should modify the actual deployment-descriptor file only if you are very familiar with XML syntax.

Note: For more information on connection configuration properties, see "Configuring Server Access for J2EE" on page 17.

Deploying the resource adapter

To deploy the WebSphere Access Resource Adapter for use with WebSphere Application Server, perform the following steps:

1. Download and install the `Connector Architecture` (technology preview) from the IBM web site, *www.ibm.com*.
2. Before you actually deploy the Access Resource Adapter, it is recommended that you shut down the application server on which you intend to deploy the resource adapter.
3. Open the WebSphere administrative console.
4. From the drop-down menus, select:

   `Console–> Wizards–> Create J2C Connection Factory`
5. In the `Wizard` dialog box that opens, give a name for the connector factory (for example, `CWConnectionFactory`) and click Next.
6. Choose:

   `Create a new J2C resource adapter`

   and click **Next**.
7. Give a name to the Access Resource Adapter (for example, `CWResourceAdapter`), browse for and install the `CWResourceAdapter.rar` file. Click **Next**.
8. In the final dialog box, review the information presented and click Finish.

   The WebSphere Application Server now deploys the resource adapter (This might take a few minutes.). If this operation completes without errors, the resource adapter has been deployed successfully.
9. In the left-hand column of the WebSphere administrative console, expand the branch:

   `Resources–> J2C Resources–> CWResourceAdapter–> J2C Connection Factory`

   Verify that the JNDI binding path for the connection factory is:

   `eis/CWConnectionFactory`
10. After you have completed the installations, restart the WebSphere Application Server so that all changes properly take effect

Making ORB libraries available

At run time, a WebSphere Access Resource Adapter requires access to the library for the IBM Java ORB. You must ensure that the Java Virtual Machine (JVM) of the application server has access to the ORB .jar file at run time. For WebSphere Application Server, the IBM Java ORB is contained in the application-server run time. Therefore, you do not need to take special steps to provide the application server with the appropriate ORB .jar file. However, if you are using any other application server and the ORB classes are *not* included in the default libraries of its JVM, you might need to copy the ORB libraries or reference them in your system class path.

Chapter 3. Using Server Access for J2EE

This chapter provides information on how to use a WebSphere Access Resource Adapter from an application component to obtain information from an ICS-managed EIS application. It provides information on how to use this Access Resource Adapter in both the managed and non-managed environments.

This chapter contains the following sections:

Managing the connection

An instance of a WebSphere Access Resource Adapter handles the opening and closing of a connection to WebSphere Interchange Server (ICS). An application component can manage this connection through the CCI interfaces that WebSphere Server Access for J2EE provides for managing a connection (see Table 9).

Table 9. Server Access for J2EE CCI implementation for managing a connection

CCI Interface	Description	Supported Methods
`ConnectionFactory`	Provides an interface for obtaining a connection to InterChange Server. Server Access for J2EE returns an instance of the `ConnectionFactory` instance that obtains a connection to ICS.	`getConnection()`, `getMetaData()`, `getRecordFactory()`, `equals()`, `getlogWriter()`, `setlogWriter()`, `hashCode()`
`Connection`	Represents an application-level handle to access the underlying physical connection to ICS. The `getConnection()` method on a `ConnectionFactory` instance returns a `Connection` instance.	`close()`, `createInteraction()`, `getMetaData()`
`ConnectionSpec`, `CwConnectionSpec`	Passes connection-specific properties to the `ConnectionFactory.getConnection()` method	`getUserName()`, `setUserName()`, `getUserPassword()`, `setUserPassword()`
`LocalTransaction`	*Server Access for J2EE does not implement this interface.*	

In Table 9, the Supported Methods column lists those interface methods that Server Access for J2EE implements. Interface methods *not* listed there fall into one of the following categories:

- Methods that throw the `NotSupportedException` exception

  In the `Connection` interface:
  - `getAutoCommit()`, `setAutoCommit()`
  - `getResultSetInfo()`
  - `getLocalTransaction()`
- Methods that have *no* functionality

All other methods in this interface are *not* implemented; that is, a call to one of these interface methods not listed does *not* generate a compile error but the method has no functionality.

The interfaces in Table 9 provide the following connection-management tasks:

- "Establishing a connection"
- "Closing the connection" on page 27

Establishing a connection

To establish a connection, the application component must obtain a `ConnectionFactory` instance, which is capable of generating a connection to an underlying EIS. The CCI interface that represents a handle to the ICS connection is called `Connection`. The way to obtain this instance depends on whether the application component is running in the managed or non-managed environment.

In a managed environment

While an Access Resource Adapter is responsible for opening and closing ICS connections, the application server determines when such activities occur. The application server creates connection pools for different connection configurations. Connection management is *not* the responsibility of the Access Resource Adapter. In a managed environment, the application component takes the following steps to acquire a connection to ICS through an Access Resource Adapter:

1. Look up the `ConnectionFactory` instance from the Java Naming and Directory Interface (JNDI) for IBM WebSphere InterChange Server.

   The JNDI name for an Access Resource Adapter is:

   ```
   eis/CWResourceAdapter
   ```

   This connection factory generates an application-level connection handle to the ICS connection.
2. Establish a connection to ICS through an Access Resource Adapter with the `ConnectionFactory.getConnection()` method.

   The `getConnection()` method returns a `Connection` instance. If a connection already exists in one of the application server's connection pools that can satisfy the connection request, the `getConnection()` method returns a handle to that connection. Otherwise, the method requests a new connection from the Access Resource Adapter.

The following code fragment shows an application component establishing a connection to ICS in the managed environment:

```
InitialContext ic = new InitialContext();

// Look up the connection factory for an Access Resource Adapter
ConnectionFactory connFactory = (Connection Factory) ic.lookup(
  "eis/CWResourceAdapter");
// Establish a connection to ICS
myConnection = connFactory.getConnection();

// code to request execution of the collaboration goes here
.....
// Close the connection
myConnection.close();
```

The following sections provide additional information about connections within the managed environment:

- "Sharing connections" on page 25

- "Providing a different Password"

Sharing connections: In a managed environment, Server Access for J2EE supports **connection sharing**. Connection sharing is usually configurable in the application server for EJB components. Other J2EE components might have their own mechanism for connection sharing. CCI provides the following connection interfaces that are involved in connection sharing:

- The `ManagedConnection` interface represents a physical connection to the EIS.

  This interface is internal to the application server and the resource adapter. It is not available to the application component. In a managed environment, these physical connections are opened and closed under the control of the application server. Server Access for J2EE implements the `CwManagedConnection` interface to represent a connection to InterChange Server.
- The `Connection` interface is a connection handle to the underlying physical connection.

  Upon receipt of a request by the application server, an Access Resource Adapter generates a connection handle for its physical connection to ICS. The application server then distributes these connection handles to application components as needed.

When the application server supports connection sharing, the application server requests multiple connection handles (`Connection` instances) for the same physical connection (the `ManagedConnection` instance). Application components are unaware that the connection handle they receive is shared. The `CwManagedConnection` implementation ensures that concurrent requests to the same physical connection are handled in a thread-safe manner.

Providing a different Password: The `ConnectionSpec` interface provides access to connection-specific properties. The implementation of the `ConnectionSpec` interface that Server Access for J2EE provides is called `CwConnectionSpec`. The application component defines how it wants to interact with Interchange Server through a `CwConnectionSpec` instance.

The configuration properties used to connect to ICS (see Table 8) are located in the deployment descriptor for the Access Resource Adapter. Most of these properties are *not* configurable at run time. However, one property that *is* configurable is the user password used for authorizing access to the InterChange Server system. To establish a connection with a password other than the one defined in the deployment descriptor, pass a `CwConnectionSpec` instance when you request the connection.

The following line of code provides the password of `cwaccess3` for the `admin` user when it requests a connection through `getConnection()`:

```
InitialContext ic = new InitialContext();
   // Look up the connection factory for
   // IBM WebSphere InterChange Server for J2EE
   CwConnectionFactory connFactory = (CwConnection Factory) ic.lookup(
      "eis/CWConnectionFactor");
   myConnection = connFactory.getConnection(
       new CwConnectionSpec("admin", "waccess3"));
```

Important: To reference the `CwConnectionSpec` class within an application component, you must ensure that the InterChange Server `specs.jar` file gets loaded. This jar file contains the `CwConnectionSpec` class, for which the application component needs to explicitly create instances to pass parameters to an Access Resource Adapter. Some possible ways to

provide the application component with access to the specs.jar file are: adding the specs.jar file to the application component's J2EE Enterprise Archive (EAR) file, or specifying this jar file in the CLASSPATH of your application server.

In a non-managed environment

In a non-managed environment, an application component is a client application; it cannot look up a pre-existing ConnectionFactory from a JNDI service. Therefore, the client application is responsible for creating its own ConnectionFactory instance. To enable the client application to create a connection factory, CCI provides the ManagedConnectionFactory interface. The implementation of this interface that Server Access for J2EE provides is called CwManagedConnectionFactory.

Note: In a managed environment, the application server handles the creation of the CwManagedConnectionFactory instance. Therefore, the application component does *not* need to use it.

The Server Access for J2EE jar file (CWResourceAdapter.jar) contains the connection-management classes and the CCI implementation. The InterChange Server product provides a sample ConnectionManager class as part of the CWResourceAdapter.jar file. The application developer can use this default implementation. However, the ConnectionManager implementation for run time must be provided by either the application developer or third-party software. Therefore, the application developer must replace this sample with the site-specific implementation of the ConnectionManager class for run time.

In a non-managed environment, the application component takes the following steps to acquire a connection to InterChange Server through an Access Resource Adapter:

1. Create an instance of the Server Access for J2EE's implementation of the ManagedConnectionFactory interface.

   The CwManagedConnectionFactory interface is a factory for instances of the InterChange Server connection factory.

2. Specify the location of the Interoperable Object Reference (.ior) file with the setIorFilename() method.

   The CwManagedConnectionFactory interface provides setIorFilename() so that the client application can specify the location of the resource adapter's .ior file. The Access Resource Adapter needs the .ior file to establish a connection to ICS.

3. Create an InterChange Server connection factory with the createConnectionFactory() method.

   The ManagedConnectionFactory interface provides the createConnectionFactory() method to create a ConnectionFactory instance. The Server Access for J2EE's implementation of createConnectionFactory() returns an InterChange Server connection factory.

4. Establish a connection to ICS through an Access Resource Adapter with the ConnectionFactory.getConnection() method.

   The getConnection() method returns a Connection instance. If a connection already exists that can satisfy the connection request, the getConnection() method returns a handle to that connection. Otherwise, the method requests a new connection from the Access Resource Adapter. The CCI interface that represents a handle to the ICS connection is called Connection.

The following code fragment shows a client application establishing a connection to ICS:

```
// Obtain a CwManagedConnectionFactory instance and specify the
// location of the .ior file
CwManagedConnectionFactory mcf = new CwManagedConnectionFactory();
mcf.setIorFilename("/crossworlds/MyServer.ior");

// Obtain a WebSphere business integration system connection factory
connFactory = (ConnectionFactory) mcf.createConnectionFactory();

// Establish a connection to ICS
myConnection = connFactory.getConnection();
```

Important: As in the managed environment, the Access Resource Adapter is responsible only for opening and closing ICS connections. Connection management is *not* the responsibility of this resource adapter. However, in a non-managed environment, there is no application server. Therefore, the client application must handle connection management, including security strategies and connection pools.

Closing the connection

The WebSphere Access Resource Adapter does *not* monitor its connection to ICS. It checks for a connection only when it receives a request. If the connection is invalid, the Access Resource Adapter attempts to establish a new one. It it cannot, the resource adapter throws an appropriate `ResourceException` exception to notify the application component.

Note: In a managed environment, the Access Resource Adapter also sends a `CONNECTION_ERROR_OCCURRED` status to the application server when it cannot reestablish a connection.

When the application component is finished using a `Connection` instance, it should explicitly close it. A call to the `close()` method does *not* close the underlying physical connection to the InterChange Server instance; it just notifies the application server that the physical connection can be closed if no other client applications are currently using it.

Sending data to an ICS-managed EIS

Server Access for J2EE implements support for an application component to send data to an EIS application that InterChange Server manages. The application component prepares an input record and requests execution of a WebSphere Business Integration Collaboration. When a WebSphere Access Resource Adapter receives an input record from the application component, the resource adapter initiates a call-triggered flow by submitting an event to a collaboration. An application component requests this call-triggered flow through the interfaces that the CCI provides to drive an interaction (see Table 10).

Table 10. Server Access for J2EE CCI implementation for executing a collaboration

CCI Interface	Description	Supported methods
`Interaction`	Allows an application component to request the initiation of a call-triggered flow. The `createInteraction()` method on a `Connection` instance returns an `Interaction` instance.	`execute()`,`close()`,`getConnection()`

Table 10. Server Access for J2EE CCI implementation for executing a collaboration (continued)

CCI Interface	Description	Supported methods
`InteractionSpec`, `CwInteractionSpec`	Used to pass interaction-specific properties in the `Interaction.execute()` method.	`getFunctionName()`, `setFunctionName()` `getInteractionVerb()`, `setInteractionVerb()`, `getCollaborationName()`, `getCollaborationPort()`, `getCollaborationVerb()`

In Table 10, the Supported Methods column lists those interface methods that Server Access for J2EE implements. Interface methods *not* listed there fall into one of the following categories:

- Methods that are *not* surfaced to application developers
- Methods that have *no* functionality

  All other methods in this interface are *not* implemented; that is, a call to one of these interface methods not listed does *not* generate a compile error but the method has no functionality.

Keep the following points in mind when requesting execution of a collaboration through an Access Resource Adapter:

- Due to existing limitations in the J2EE Connector Architecture specification 1.0., an Access Resource Adapter can only support unidirectional call-triggered flow.

  An Access Resource Adapter supports only access-request operations, which are events that the application component initiates and needs to send to some EIS application. This resource adapter does *not* support event notification; that is, it cannot support events that some other WebSphere business integration system component has initiated.
- Because this initiated flow is a call-triggered flow, the event is processed synchronously and is *not* persistent within the InterChange Server system.

  If ICS unexpectedly terminates during execution of the call-triggered flow, the application component is *not* notified of the failure through an exception. However, the Access Resource Adapter would be informed whether the request could be processed *before* the termination. To recover:

  – For retrieval requests, the application component can reissue the retrieval request.

  – For requests that modify data in an ICS-managed EIS, the application component must determine whether the request was completed once ICS has been restarted.

  For more information on call-triggered flow, see the *Access Development Guide.*

Initiating a call-triggered flow in the application component involves the following steps:

- "Identifying the collaboration to execute"
- "Creating the serialized input record" on page 30
- "Invoking the execute() method" on page 31
- "Closing the interaction" on page 31

Identifying the collaboration to execute

The `InteractionSpec` interface provides interaction-specific properties. The implementation of the `InteractionSpec` interface that Server Access for J2EE

provides is called CwInteractionSpec. The application component defines how it wants to interact with InterChange Server through a CwInteractionSpec instance.

Table 11 shows the standard interaction properties of the InteractionSpec interface. As Table 11 shows, the CwInteractionSpec interface supports two of these standard properties.

Table 11. CwInteractionSpec standard interaction properties

Standard interaction property	Description	WebSphere business integration system support
FunctionName	The collaboration to send to InterChange Server for execution	Server Access for J2EE expects the FunctionName to contain the following collaboration routing information: • the name of the collaboration to initiate • the port of the collaboration that receives the input record • the WebSphere business integration system verb for the collaboration (*optional*) The application component can concatenate the name and port information into one delimited string and the resource adapter can parse it.
InteractionVerb	An integer that represents the mode of interaction with InterChange Server. The InteractionSpec defines the following constants for these integers: • SYNC_SEND • SYNC_SEND_RECEIVE • SYNC_RECEIVE	Server Access for J2EE supports *only* the default interaction verb: • SYNC_SEND_RECEIVE: The execution of an Interaction instance requests execution of the collaboration and receives a response synchronously. Use of this verb indicates that the resource adapter both sends and receives data in a single, blocking call. Because SYNC_SEND_RECEIVE is the default interaction verb, you do not need to explicitly set InteractionVerb to work with Server Access for J2EE. If you set the interaction verb to anything other than this default, the setInteractionVerb() method throws the exception PropertyVetoException.
ExecutionTimeout	The number of milliseconds an Interaction instance waits for the EIS to execute the specified function	*Server Access for J2EE does not support this standard property.*

Note: Because InteractionSpec is implemented as a Java Bean, the CwInteractionSpec interface supports get and set methods to access the interaction properties (see Table 10).

The first step in requesting a call-triggered flow is to identify the collaboration that needs to execute. To identify the collaboration, specify the collaboration name, port, and verb in the CwInteractionSpec instance. Use the following syntax to set the FunctionName standard property to include the name and port of the collaboration, as well as the verb:

CollaborationName:*Port*:*CollaborationVerb*

As the syntax above indicates, a colon (:) separates each of these pieces of information.

Note: The collaboration and port that you specify must be configured for call-triggered flow. In particular, the port must be bound as an external port. For more information on how to configure a collaboration for a call-triggered flow, see the *Access Development Guide*.

You can set the FunctionName standard property in either of the following ways:

- Pass the FunctionName value to the InteractionSpec() constructor.

  The following line of code identifies the collaboration based on the initial value provided to the CwInteractionSpec() constructor:

  ```
  iSpec = new CwInteractionSpec("Customer:NewCust:Create");
  ```

  This initial value specifies initiation of the Customer collaboration, which creates a new customer from the business object it receives on its NewCust port.

  Note: The presence of a WebSphere business integration system verb in the FunctionName string is optional. If you omit the verb from FunctionName, you *must* explicitly specify it in the instance of the input MappedRecord instance. For more information, see "CwMappedRecord" on page 33.

- Use the setFunctionName() method.

  The following lines of code identify the collaboration based on the argument passed to the setFunctionName() method:

  ```
  iSpec = new CwInteractionSpec();
  iSpec.setFunctionName("Customer:NewCust:Create");
  ```

  This argument in setFunctionName() specifies initiation of the Customer collaboration, which creates a new customer from the business object it receives on its NewCust port.

Important: To reference the CwInteractionSpec class within an application component, you must ensure that the InterChange Server specs.jar file gets loaded. This jar file contains the CwInteractionSpec class, which the application component needs to explicitly create instances of to pass parameters to an Access Resource Adapter. Some possible ways to provide the application component with access to the specs.jar file are: adding the specs.jar file to the application component's J2EE Enterprise Archive (EAR) file, or specifying this jar file in the CLASSPATH of your application server.

Creating the serialized input record

The application component stores the data to send to an EIS in an input record. The WebSphere Access Resource Adapter converts this input record into a business object, which is the triggering event for the collaboration. The Server Access for J2EE's implementation of the MappedRecord interface, CwMappedRecord, holds this serialized input record, which the application component then passes to the Interaction.execute() method.

Note: The Server Access for J2EE's implementation of the Interaction interface accepts *only* CwMappedRecord instances for the execute() method. However, a CwMappedRecord instance can contain CwIndexedRecord instances and other CwMappedRecord instances.

If the verb for the collaboration is *not* specified in the InteractionSpec instance, the resource adapter extracts the verb from the top-level CwMappedRecord that is passed into the execute() method.

For more information on CwMappedRecord and CwIndexedRecord and how to create them, see "Using records" on page 32.

Invoking the execute() method

The CCI method that actually sends the request to initiate the call-triggered flow is Interaction.execute(). The implementation of execute() that Server Access for J2EE provides takes the following steps:

1. Convert the input record to a business object.
2. Initiate a call-triggered flow by sending the triggering event to the specified collaboration.

To call the execute() method, the application component must take the following steps:

1. Generate an Interaction instance with the Connection.createInteraction() method.
2. Invoke the execute() method on the Interaction instance to initiate the call-triggered flow by requesting execution of a collaboration.

   Server Access for J2EE supports the following two forms of the Interaction.execute() method:

   - One form of execute() requests the execution of the collaboration that the InteractionSpec instance identifies and provides the input record as the triggering event. This form converts the business object received from the collaboration to the output record that is provided as an argument. The syntax for this first form of execute() is as follows:

     execute(*InteractionSpec*, *input_record*, *output_record*)

   - The second form of execute() also requests the execution of the collaboration that the InteractionSpec instance identifies and provides the input record as the triggering event. However, this form converts the business object received from the collaboration to the return value of execute(). The syntax for this second form of execute() is as follows:

     output_record = execute(*InteractionSpec*, *input_record*)

The following pseudo-code shows how an application component can request information about an employee named "Stan Smith":

```
// Obtain the input record (a CwMappedRecord instance) and populate it
inputRecord =
  connFactory.getRecordFactory().createMappedRecord("Employee");
inputRecord.put("FirstName", "Stan");
inputRecord.put("LastName", "Smith");

// Set the interaction-specific properties in the CwInteractionSpec
iSpec = new CwInteractionSpec("myCollab:fromObject:Retrieve");

// Generate the Interaction instance and then execute it
myInteraction = myConnection.createInteraction();
outputRecord = myInteraction.execute(iSpec, inputRecord);

// Obtain the data from the output record
employeeId = outputRecord.get("EmployeeId");

// Close the interaction
myInteraction.close();
```

Closing the interaction

When the application component is finished using an Interaction instance, it should explicitly close it. A call to the close() method releases all resources that

the WebSphere Access Resource Adapter has maintained for the Interaction. The close of an Interaction instance does *not* close the associated connection handle. It is recommended that you close Interaction instances explicitly to free any held resources.

Using records

The WebSphere Access Resource Adapter sends information to and receives information from a collaboration as a WebSphere business integration system business object. An application component sends information to and receives information from the Access Resource Adapter through the Java interfaces for the Record type (see Table 12).

Table 12. Server Access for J2EE CCI implementation for creating a record

CCI Interface	Description	Supported methods
RecordFactory	Allows an application component to create an instance of a Record extension. The getRecordFactory() method on a ConnectionFactory instance returns a RecordFactory instance.	createMappedRecord() createIndexedRecord()
CwMappedRecord	The Record extension that stores data in key-value pairs. It represents a single-cardinality business object.	All methods of the Record and Map interfaces are supported.
CwIndexedRecord	The Record extension that stores data as an ordered, indexed list. It represents a multiple-cardinality attribute.	All methods of the Record and Map interfaces are supported.

In Table 12, the Supported Methods column lists those interface methods that Server Access for J2EE implements. Interface methods *not* listed there fall into one of the following categories:

- Methods that are *not* surfaced to application developers
- Methods that have *no* functionality

  All other methods in this interface are *not* implemented; that is, a call to one of these interface methods not listed does *not* generate a compile error but the method has no functionality.

This section provides the following information about how to handle Record extensions:

- "Using extensions to record"
- "Creating a record" on page 34

Using extensions to record

A Record is the Java representation of a data structure that encapsulates the data exchanged between an application component and the underlying EIS. It is used as input and output in the Interaction.execute() method. Server Access for J2EE supports the Record extensions shown in Table 13.

Table 13. Record extensions that Server Access for J2EE supports

Record extension	Server Access for J2EE class	Description
MappedRecord	CwMappedRecord	Based on the Java interface:java.util.Map
IndexedRecord	CwIndexedRecord	Based on the Java interface:java.util.List

Note: Server Access for J2EE does *not* support the ResultSet extension to the Record interface.

The MappedRecord and IndexedRecord interfaces are hierarchical; that is, they can contain other Record structures.

CwMappedRecord

To pass business data to a WebSphere Access Resource Adapter, an application component needs to create a CwMappedRecord instance and pass it to the Interaction.execute() method as an input record. A CwMappedRecord instance represents a WebSphere business integration system business object, which is an instance of the WebSphere business integration system BusObj class. It holds one or more key-value pairs in the same way that a BusObj instance holds one or more pairs of attributes and their associated values. Therefore, each key in a CwMappedRecord instance must contain a key-value pair for each of the following:

- The name of the business object
- The WebSphere business integration system verb (if *not* specified in the InteractionSpec instance)
- Each attribute in the intended business object

  Make sure that you match the names of the business object attributes in the key-value pairs of the CwMappedRecord instance. The Access Resource Adapter does *not* throw an exception if these names do not match. Any mismatch is not detected until the data is passed to InterChange Server during the execution of the interaction.

Important: All CwMappedRecord keys must be of type java.lang.String. No other data type is valid for a key. Even if your key value is of type Integer, you must still specify the string representation instead.

Figure 9 shows the correspondence between a CwMappedRecord instance and a WebSphere business integration system business object (BusObj instance).

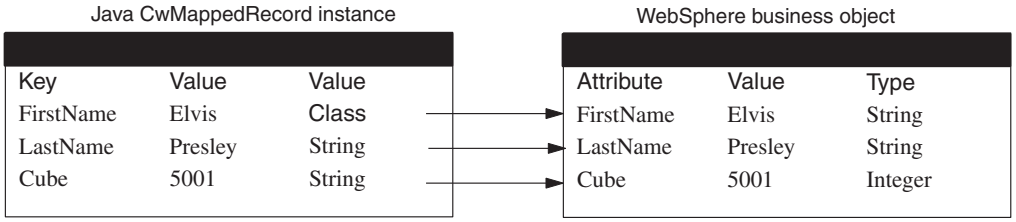

Figure 9. Correspondence between CwMappedRecord and a business object

To specify a single-cardinality child object, set the value of the appropriate key to another CwMappedRecord instance. To specify a multiple-cardinality child object, set the value of the appropriate key to a CwIndexedRecord instance. For more information, see "CwIndexedRecord" on page 33.

CwIndexedRecord

A Java IndexedRecord represents an ordered and indexed list. A CwIndexedRecord instance represents an array of WebSphere business integration system business objects, which is an instance of the WebSphere business integration system BusObjArray class. It holds an n-cardinal list of elements in the same way that a BusObjArray instance holds a multiple-cardinality attribute (and its values). Therefore, each element in a CwIndexedRecord instance must correspond to a business object.

Figure 10 shows the correspondence between a `CwIndexedRecord` instance and a WebSphere business integration system business object array (`BusObjArray` instance).

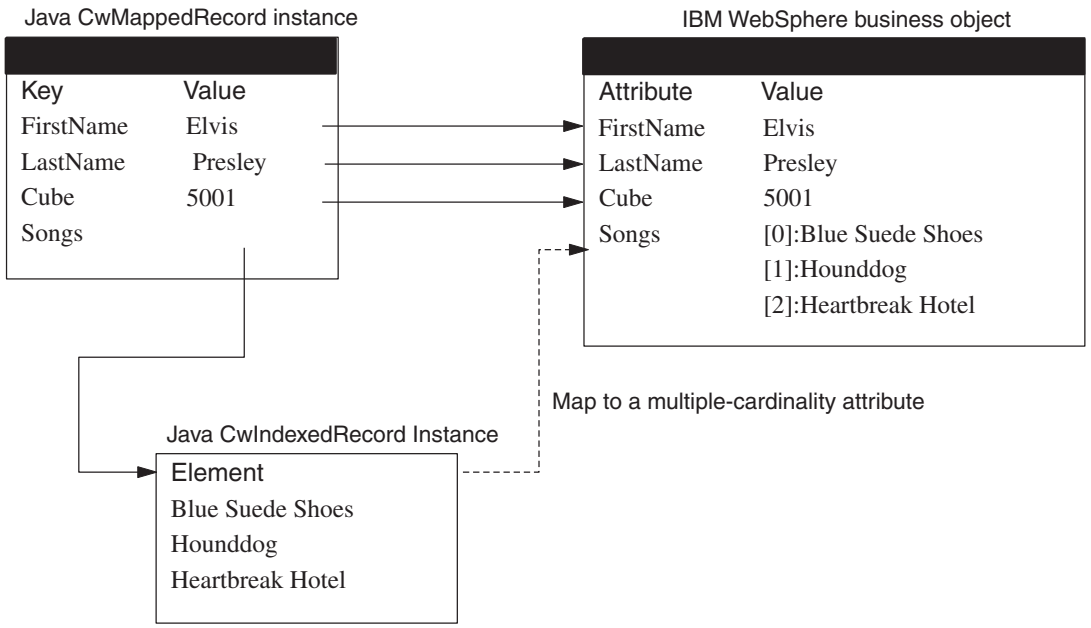

Figure 10. Correspondence between CwIndexedRecord and a business object array

The `CwIndexedRecord` contains one or more elements, each of which can contain a primitive value (such as the `String` values in Figure 10) or a `CwMappedRecord` instance. However, *all* elements must have the same type.

Creating a record

The WebSphere Access Resource Adapter converts between `Record` instances and business objects. It communicates with the application component through `Record` instances and with InterChange Server through business objects. Table 14 shows the business objects involved in the execution of a collaboration along with their equivalents within the application component.

Table 14. Business objects involved in collaboration execution

In a collaboration	In the execute() method
Triggering business object	Input record
Business object returned by the collaboration	Output record

To provide data to an InterChange Server-managed EIS, an application component must perform the following tasks with records:

- Create an input record, which the Access Resource Adapter converts to the triggering business object.
- Create an output record, which the Access Resource Adapter converts from the business object that the collaboration populates (*optional*).

To create a `Record` extension to send to an Access Resource Adapter, the application component must take the following steps:

1. Use the `ConnectionFactory.getRecordFactory()` method to obtain a `RecordFactory` object.

The record factory is capable of generating Record instances.

2. Use the appropriate RecordFactory method to obtain the desired Record extension:
   - For a CwMappedRecord, use the createMappedRecord() method.

     This method requires as an argument the name of the record to be created. This argument must have the following syntax:

     <bus_obj>:<verb>

     where bus_obj is the name of the business object and *verb* is the name of the WebSphere business integration system verb. Inclusion of the *verb* is optional.
   - For a CwIndexedRecord, use the createIndexedRecord() method.

     This method requires as an argument either an empty string ("") or the keyword null.

Obtaining metadata

Server Access for J2EE provides additional information about an ICS connection and about the WebSphere Access Resource Adapter itself. An application component can access this information through the interfaces of the CCI that provide metadata. In Table 15, the Supported Methods column lists those interface methods that Server Access for J2EE implements.

Table 15. Server Access for J2EE CCI Implementation for Obtaining Metadata

CCI Interface	Description	Supported Methods
ConnectionMetaData, CwConnectionMetaData	Provides information about the connection to WebSphere Interchange Server. The getMetaData() method on a Connection instance returns a ConnectionMetaData instance.	All methods shown in Table 16.
ResourceAdapterMetaData, CwResourceAdapterMetaData	Provides information about the WebSphere Access Resource Adapter. The getMetaData() method on a ConnectionFactory instance returns a ResourceAdapterMetaData instance.	All methods shown in Table 17.

An application component can perform the following tasks to obtain metadata:

- "Obtaining metadata for a connection"
- "Obtaining metadata for the Access Resource Adapter" on page 36

Obtaining metadata for a connection

The ConnectionMetaData interface provides information about the connection to the EIS. The implementation of the ConnectionMetaData interface that Server Access for J2EE provides is called CwConnectionMetaData, which allows an application component to obtain information about a connection it has established to InterChange Server through a Connection instance. Table 16 shows the methods that the ConnectionMetaData interface provides for obtaining metadata for a connection. The CwConnectionMetaData interface implements *all* methods in Table 16.

Table 16. Methods in the ConnectionMetaData interface

ConnectionMetaData method	Description
getEISProductName()	WebSphere InterChange Server

Table 16. Methods in the ConnectionMetaData interface (continued)

ConnectionMetaData method	Description
`getEISProductVersion()`	The version of the InterChange Server software, usually of the form: x.x.x
`getUserName()`	A user-defined value that is retrieved from the configuration properties

To obtain connection metadata, the application component takes the following steps:

1. Use the `Connection.getMetaData()` method to obtain a `ConnectionMetaData` object.
2. Use the appropriate `ConnectionMetaData` method (see Table 16) to obtain the metadata from this object.

Obtaining metadata for the Access Resource Adapter

The `ResourceAdapterMetaData` interface provides information about the WebSphere Access Resource Adapter. The implementation of the `ResourceAdapterMetaData` interface that Server Access for J2EE provides is called `CwResourceAdapterMetaData`, which allows an application component to obtain information about the capabilities of the Access Resource Adapter. Table 17 shows the methods that the `ResourceAdapterMetaData` interface provides for obtaining metadata for the resource adapter. The `CwResourceAdapterMetaData` interface implements all methods in Table 17.

Table 17. Methods in the ResourceAdapterMetaData interface

ResourceAdapterMetaData method	WebSphere business integration system return value
`getAdapterName()`	`IBM Crossworlds Resource Adapter for InterChange Server`
`getAdapterShortDescription()`	`IBM Crossworlds Resource Adapter for InterChange Server`
`getAdapterVendorName()`	`IBM Corp.`
`getAdapterVersion()`	`1.1.0`
`getInteractionSpecsSupported()`	An array with one element:`com.crossworlds.adapter.CwInteractionSpec`
`getSpecVersion()`	`1.0`
`supportsExecuteWithInputAndOutputRecord()`	`true`
`supportsExecuteWithInputRecordOnly()`	`true`
`supportsLocalTransactionDemarcation()`	`false`

Note: In Table 17, the return value of `true` for the methods `supportsExecuteWithInputandOutputRecord()` and `supportsExecuteWithInputRecordOnly()` indicates that Server Access for J2EE supports both forms of the `Interaction.execute()` method. For more information, see "Invoking the execute() method" on page 31.

To obtain metadata for the Access Resource Adapter, the application component takes the following steps:

1. Use the `ConnectionFactory.getMetaData()` method to obtain a `ResourceAdapterMetaData` object.
2. Use the appropriate `ResourceAdapterMetaData` method (see Table 17) to obtain the metadata from this object.

For example, the following lines determine whether Server Access for J2EE supports the first form of the `Interaction.execute()` method (the form that supports both input and output records):

```
res_adpt_metadata = connFactory.getMetaData();
if (res_adpt_metadata.supportsExecuteWithInputandOutputRecord())
  {
  // takes steps to initialize input record

  // call first form of execute()
  Interaction.execute(.....);
```

Handling exceptions

Server Access for J2EE provides the following features for handling exceptions:

- "Logging and tracing"
- "Exceptions" on page 38

Logging and tracing

Server Access for J2EE implements necessary error and tracing interfaces.

In a managed environment

In a managed environment, the application server handles all messages that the WebSphere Access Resource Adapter generates in its own server-specific manner. Usually, you can configure this structure through a console or a deployment descriptor.

Note: This section summarizes logging and tracing at a high level. For more detailed information, refer to the documentation for your application server.

Debugging a resource adapter is largely dependent on the application server in which the resource adapter is deployed. An Access Resource Adapter handles logging and tracing as follows:

- Logging

  An Access Resource Adapter does *not* use the logging facilities provided in the WebSphere business integration system connector infrastructure. Instead, the resource adapter uses the logging and tracing services that the application server provides. Therefore, there is no concept of a message file for an Access Resource Adapter. The application server logs all messages that the resource adapter generates in its own server-specific manner. Usually, you can configure this structure through a console or a deployment descriptor. For more information, refer to the documentation for your application server.

- Tracing

  As with logging, the Access Resource Adapter uses the tracing services that the application server provides. Therefore, there is no concept of a trace level for an Access Resource Adapter. The application server can support tracing for connection management, as Server Access for J2EE does implement necessary error and tracing interfaces.

In a non-managed environment

In a non-managed environment, the client application that calls the WebSphere Access Resource Adapter must handle all error logging and tracing.

Exceptions

Server Access for J2EE provides support for exception handling. An application component checks for exceptions that the InterChange Server connection generates through the CCI classes shown in Table 18.

Table 18. CCI classes for exception handling

CCI class	Description
`ResourceException`	Thrown by a WebSphere Access Resource Adapter to indicate an exception has occurred in any of the following cases: • The processing of data • The connection or interaction with InterChange Server.
`ResourceWarning`	*Server Access for J2EE does not support this class.*

Important: The J2EE Connector Architecture specification 1.0. provides warnings for the resource adapter to log noncritical errors as it interacts with the EIS. However, Server Access for J2EE does *not* support warnings. Only the default behavior of the `Interaction.getWarnings()` method is implemented; that is, this method returns `null`.

Table 19 shows the types of exceptions that an Access Resource Adapter throws.

Table 19. Types of ResourceException Exceptions

Kind of Exception	Description	Handled By
Application exception	An exception that occurs while the collaboration executes or ICS is accessed	Application component
System exception	An exception not meant to be surfaced to the application component	Application system

The exceptions in Table 19 are derived from `ResourceException`, which extends `Exception`. A `ResourceException` provides the following information:

- A resource-adapter-specific string that describes the error

  This string is a standard Java exception message and is available through the `getMessage()` method.
- A resource-adapter-specific error code that identifies the error

  This error code identifies the error condition that the `ResourceException` instance represents.
- A reference to another exception

  Often a `ResourceException` results from a problem at a lower level. If appropriate, a lower-level exception might be linked to a `ResourceException` instance. This lower-level exception is a `java.lang.Exception` or any derived exception type. For example, an exception thrown by any method of the Server Access Interface is propagated as an `ApplicationException`, which is linked to a `ResourceException`.

Nearly all CCI methods throw a `ResourceException` if an error occurs. The application component should catch the `ResourceException` and take appropriate recovery action. Only methods in the `CwInteractionSpec` interface do *not* throw a `ResourceException`. Instead, the set methods in this interface throw the `PropertyVetoException` exception.

Notices

IBM may not offer the products, services, or features discussed in this document in all countries. Consult your local IBM representative for information on the products and services currently available in your area. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any non-IBM product, program, or service.

IBM may have patents or pending patent applications covering subject matter described in this document. The furnishing of this document does not give you any license to these patents. You can send license inquiries, in writing, to:

IBM Director of Licensing
IBM Corporation
North Castle Drive
Armonk, NY 10504-1785
U.S.A.

The following paragraph does not apply to the United Kingdom or any other country where such provisions are inconsistent with local law:

INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Some states do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement may not apply to you.

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make improvements and/or changes in the product(s) and/or program(s) described in this publication at any time without notice.

Any references in this information to non-IBM Web sites are provided for convenience only and do not in any manner serve as an endorsement of those Web sites. The materials at those Web sites are not part of the materials for this IBM product and use of those Web sites is at your own risk.

IBM may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation to you.

Licensees of this program who wish to have information about it for the purpose of enabling: (i) the exchange of information between independently created programs and other programs (including this one) and (ii) the mutual use of the information which has been exchanged, should contact:

IBM Burlingame Laboratory Director
IBM Burlingame Laboratory
577 Airport Blvd., Suite 800

Burlingame, CA 94010
U.S.A

Such information may be available, subject to appropriate terms and conditions, including in some cases, payment of a fee.

The licensed program described in this document and all licensed material available for it are provided by IBM under terms of the IBM Customer Agreement, IBM International Program License Agreement, or any equivalent agreement between us.

Any performance data contained herein was determined in a controlled environment. Therefore, the results obtained in other operating environments may vary significantly. Some measurements may have been made on development-level systems and there is no guarantee that these measurements will be the same on generally available systems. Furthermore, some measurement may have been estimated through extrapolation. Actual results may vary. Users of this document should verify the applicable data for their specific environment.

Information concerning non-IBM products was obtained from the suppliers of those products, their published announcements or other publicly available sources. IBM has not necessarily 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.

This information may contain 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.

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

Programming interface information

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

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

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

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

Trademarks and service marks

The following terms are trademarks or registered trademarks of International Business Machines Corporation in the United States or other countries, or both:

IBM
the IBM logo
AIX
CICS
CrossWorlds
DB2
DB2 Universal Database
Domino
IMS
Informix
iSeries
Lotus
Lotus Notes
MQIntegrator
MQSeries
MVS
OS/400
Passport Advantage
SupportPac
WebSphere
z/OS

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

MMX, Pentium, and ProShare are trademarks or registered trademarks of Intel Corporation in the United States, other countries, or both.

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

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

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

IBM WebSphere InterChange Server v4.3, IBM WebSphere Business Integration Toolset v4.3

Index

A

B

C

D

E

F

G

H

I

J

L

M

N

O

P

R

S

T

U

W

Printed in USA