IBM WebSphere InterChange Server

Access Development Guide for Enterprise JavaBeans TM

Version 4.3.0

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

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 IBM WebSphere InterChange Server Access for Enterprise JavaBeans (EJB) to access EIS applications that are managed through InterChange Server.

Audience

This document is for IBM WebSphere customers, consultants, or resellers who integrate J2EE application components with the WebSphere business integration system by using the Access Framework for Enterprise JavaBeans. 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 calls to the home and remote interfaces in the client component, you should know the standard programming concepts and practices as well as the Java programming language. In addition, you need to follow standards as outlined in the Enterprise JavaBeans specification 1.1.

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 Server Access for Enterprise JavaBeans uses the IBM WebSphere InterChange Server Access to communicate with InterChange Server. For information on the Server Access Interface, see the *Access Development Guide*. While this manual is not a prerequisite for a WebSphere business integration system solution to communicate with InterChange Server, knowledge of features in the InterChange Server Access can be helpful.
- A J2EE application component accesses an EIS through 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.

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.
UNIX:/Windows:	Paragraphs beginning with either of these indicate notes listing operating system differences.
u	This symbol indicates the end of a **UNIX/Windows** paragraph; it can also indicate the end of a multiparagraph note.
`%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 the Access Development Guide for Enterprise JavaBeans^{TM} for the IBM WebSphere InterChange Server development environment.

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 release 4.2.2

February 2004

For the 4.2.2 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 enterprise beans and WebSphere Server Access for Enterprise JavaBeans. For more information, see Chapter 1, "Introduction to Enterprise JavaBeans technology," 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 26.

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 EJB now uses the IBM Java Object Request Broker (ORB) to handle communication between the WebSphere Access Resource Adapter and InterChange Server.

New in release 4.2.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 "IBM WebSphere InterChange Server."

New in release 4.1.1

This product has been internationalized. The `executeCollaborationExtendedWithLocale()` method enables a developer to execute a collaboration with a specific locale.

New in release 4.1.0

The changes made in IBM WebSphere InterChange Server version 4.1.0 do not affect the content of this document.

Chapter 1. Introduction to Enterprise JavaBeans technology

This chapter provides an introduction to the support that IBM WebSphere products provide for the Enterprise JavaBeans (EJB) technology. This chapter contains the following sections:

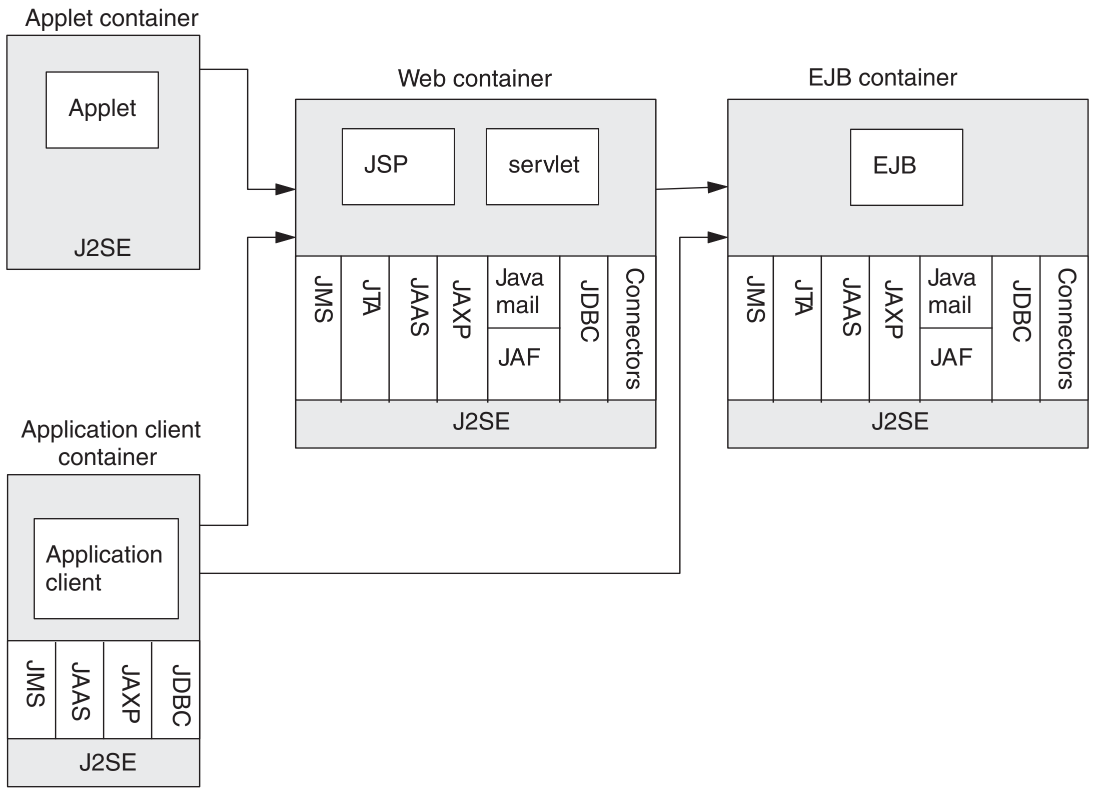
- "Overview of J2EE architecture"
- "Overview of EJB technology" on page 2
- "Overview of client-component development" on page 15
- "Processing locale-dependent data" on page 17

Overview of J2EE architecture

Java 2 Enterprise Edition (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, v1.3.



J2SE = Java 2 Platform, Standard Edition	JDBC = Java Database Connectivity
JAAS = Java Authentication and Authorization Service	JMS = Java Message Service
JAF = JavaBeans Activation Framework	JSP = Java Server Page
JAXP = Java API for XML Parsing	JTA = Java Transaction API

Figure 1. Logical relationships of the J2EE architecture

Overview of EJB technology

The EJB technology is the server-side component architecture for J2EE. The EJB technology is emerging as the technology for the development and deployment of business logic within a larger enterprise application. It is predominantly used in the middle tier of an N-tier architecture. This middle tier provides communication between client components of the client tier and Enterprise Information Systems (EISs) of the server tier, as Figure 2 shows.

Note: This section provides a high-level overview of the Enterprise JavaBeans specification 1.1. For a more complete discussion, see the Sun Java Web site at *http://java.sun.com/products/ejb/docs.html*.

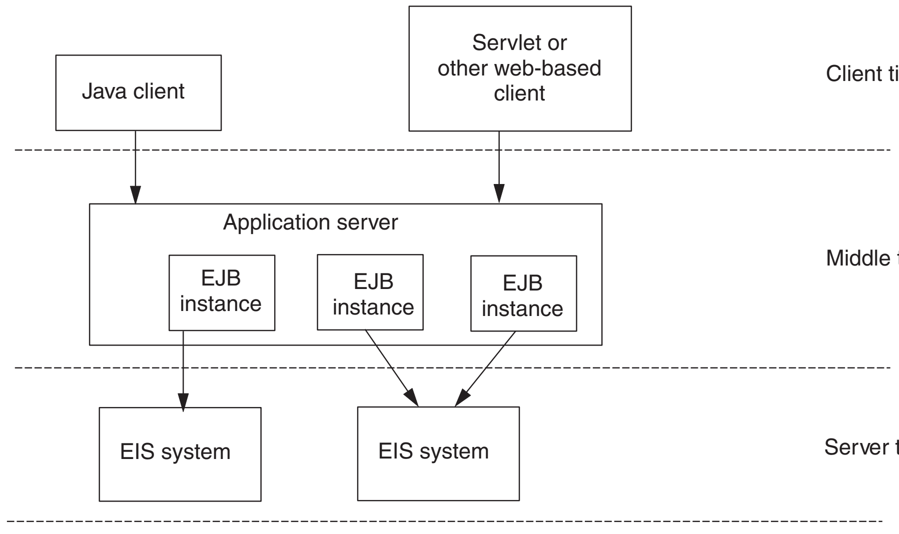


Figure 2. Example of N-tier architecture

The EJB technology allows users to isolate their business logic in the middle tier, away from the actual presentation and data layers (the client and server tiers, respectively). As Figure 2 shows, this middle tier is made up of the following components:

- An application server
- Instances of Enterprise JavaBeans, called simply enterprise beans (or EJBs)

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

Application server

In the context of an enterprise bean, an **application server** provides basic resource-allocation services to enterprise beans, which access EISs. It provides support for many middleware services, including the following:

- Connection management
- Security
- Transaction management

The benefit of accessing an EIS through an application server is that the client component does not need to know the details of connection management, security management, and transaction management. As Figure 3 shows, the **client component** is a client-side module that communicates with an application server to access various components (such as EJBs) that the application server manages.

An application server interacts with an EJB instance through an **EJB container** (see Figure 3). This container is a Java component that the application server implements. It manages the execution of the EJB instance by providing its run-time environment. In most cases, the same vendor provides both the application server and an implementation of an EJB container that executes within the application server.

Though the EJB container manages interactions between the EJB instance and the client component, it is *not* visible to the client component. Instead, the client component communicates with the EJB instance through a pair of interfaces listed in Table 1.

Table 1. EJB interfaces for client component access to an EJB

EJB interface for client component	Description	EJB interface
Home interface	Used by the client component to create and remove an EJB instance.	`EJBHome`
Remote interface	Used by the client component to request execution of a business method of the enterprise bean. A **business method** encapsulates particular business logic that the client component needs the EJB to perform.	`EJBObject`

Figure 3 shows the EJB architecture, which uses an EJB container within an application server to manage communication between a client component and an EJB instance.

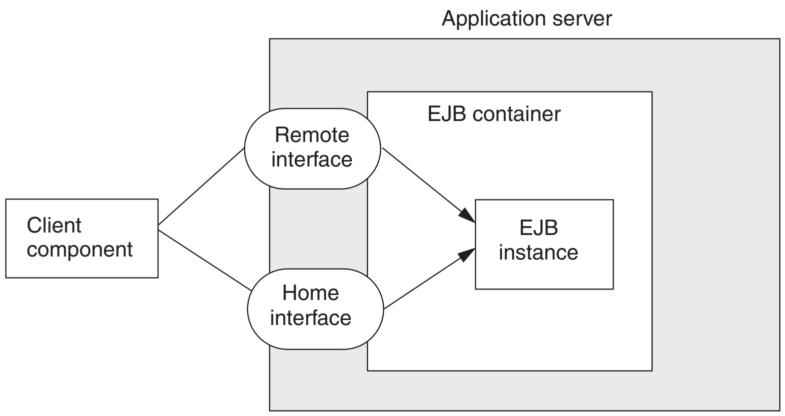


Figure 3. Accessing an EIS through an EJB

Enterprise bean

An enterprise bean is a server-side component in the J2EE architecture. It enables client components to communicate with an EIS. The client component sends a request for information in an EIS through the enterprise bean. The enterprise bean communicates directly with the EIS, returning any requested information to the client component. The client component and enterprise bean communicate with each other through the home and remote interfaces (see Figure 3), which are implemented by the EJB container. The Enterprise JavaBeans specification 1.1 provides standards for these home and remote interfaces.

The Enterprise JavaBeans specification 1.1 also defines two kinds of enterprise beans, each of which is implemented as a particular Java interface, as Table 2 shows.

Table 2. Types of enterprise beans

Type of enterprise bean	Description	EJB interface
Entity bean	Represents a business concept; usually is persistent.	`EntityBean`
Session bean	Responsible for managing a process or task; usually exists only for the duration of a session.	`SessionBean`

Note: Because Server Access for EJB implements its enterprise bean as a session bean, the remainder of this section concentrates on describing a session bean rather than an entity bean. For more information on an entity bean, consult the Enterprise JavaBeans specification 1.1.

A session bean can be one of two types:
- Stateful session bean—maintains state information, which can be accessed across methods and transactions.
- Stateless session bean—does not maintain a state that can be accessed across methods and transactions; however, it can maintain an internal state.

A session bean must provide the following information to an application server for the bean's deployment within an EJB container:
- Definitions of the session bean's home and remote interfaces
- A Java class that implements the `SessionBean` interface
- A deployment descriptor called `ejb-jar.xml`

Home and remote interfaces

The home and remote interfaces provide the EJB methods that are externalized to client components, as follows:
- The **home interface** provides methods that the client component uses to manage the EJB instance, including:
  - One or more `create()` methods that create a new EJB instance
  - A `remove()` method that removes an existing EJB instance

  The `EJBHome` interface defines the home interface.
- The **remote interface** includes methods that the client component uses to interact with the EJB instance; these methods are called the business methods of the EJB.

  The `EJBObject` interface defines the remote interface.

The EJB provider must provide classes that define the home and remote interfaces of the EJB. When the enterprise bean is deployed, the deployment tools of the application server use these definitions to generate the implementations of the home and remote interfaces for the EJB container. The client component uses these implementations when it originates a request for the enterprise bean. In this way, all interactions with the enterprise bean go through the EJB container, which routes them to the enterprise bean.

Note: In Java, an interface provides a specification of the behavior of an object but not the actual behavior. There is an important distinction between *defining* an interface (which provides only the names of the interface and its methods) and implementing an interface (which provides the actual code to implement the interface methods). The EJB provider provides the *definitions* of the home and remote interfaces while the EJB container provides the *implementations* of these interfaces using information in the deployment descriptor.

Figure 4 shows how information is provided for the home and remote interfaces of a sample enterprise bean called `sessionBean`. This enterprise bean provides two `create()`methods in its home interface; the client component can create an instance of this session bean with either of these methods. The bean also provides two business methods in its remote interface; the client component can interact with this session-bean instance with these two business methods.

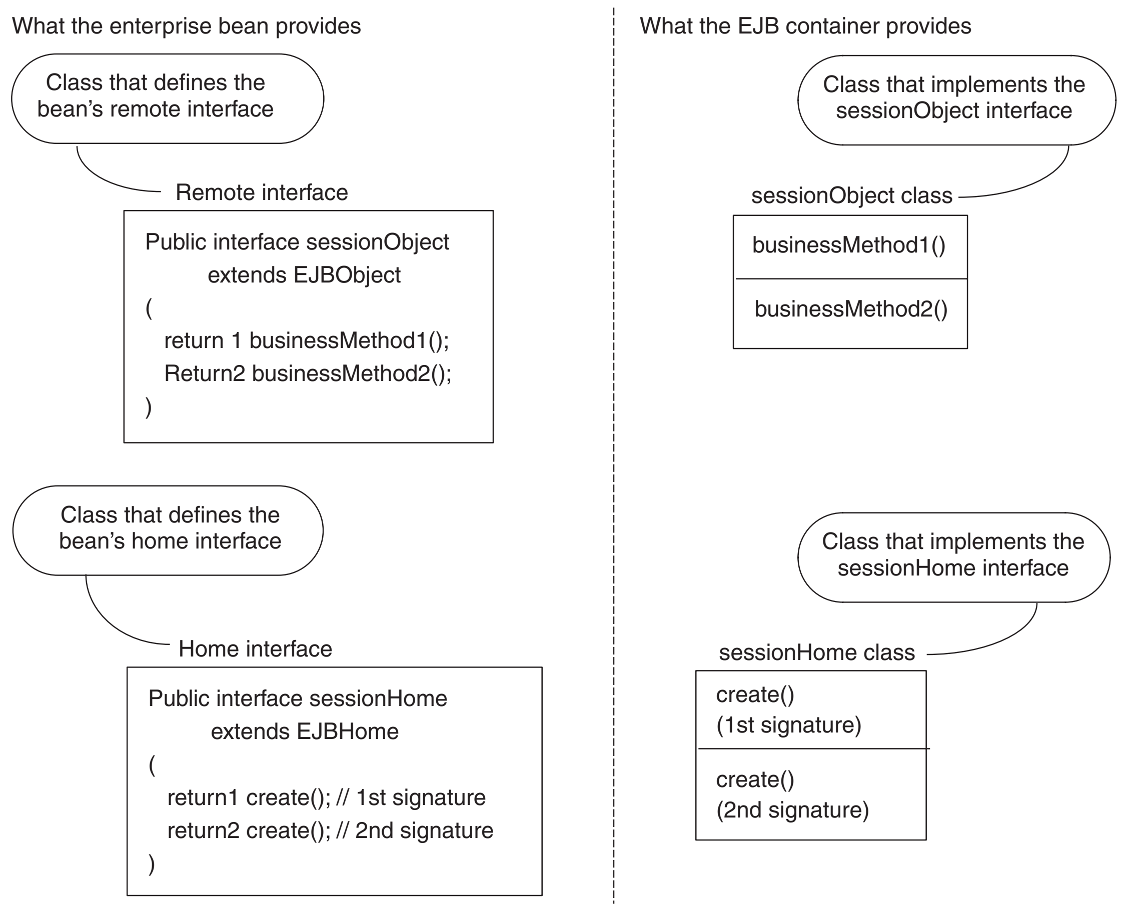

What the enterprise bean provides
Class that defines the bean's remote interface
Remote interface
Public interface sessionObject extends EJBObject (return 1 businessMethod1(); Return2 businessMethod2();)
Class that defines the bean's home interface
Home interface
Public interface sessionHome extends EJBHome (return1 create(); // 1st signature return2 create(); // 2nd signature)
What the EJB container provides
Class that implements the sessionObject interface
sessionObject class
businessMethod1()
businessMethod2()
Class that implements the sessionHome interface
sessionHome class
create() (1st signature)
create() (2nd signature)

Figure 4. Providing the home and remote interfaces of a session bean

SessionBean class

For an EJB container to communicate with the session bean, an EJB provider must provide a Java class that implements the `SessionBean` interface. This class contains implementations of the following methods:

- An `ejbCreate()` method for each `create()` method in the home interface
- One or more business methods for the session bean
- Other standard methods of the `SessionBean` interface
- Possible additional methods required to support the implementation of the `SessionBean` class

When the EJB container implements the methods of the home and remote interface, it includes in these methods calls to the corresponding methods of the `SessionBean` class, as Figure 5 shows.

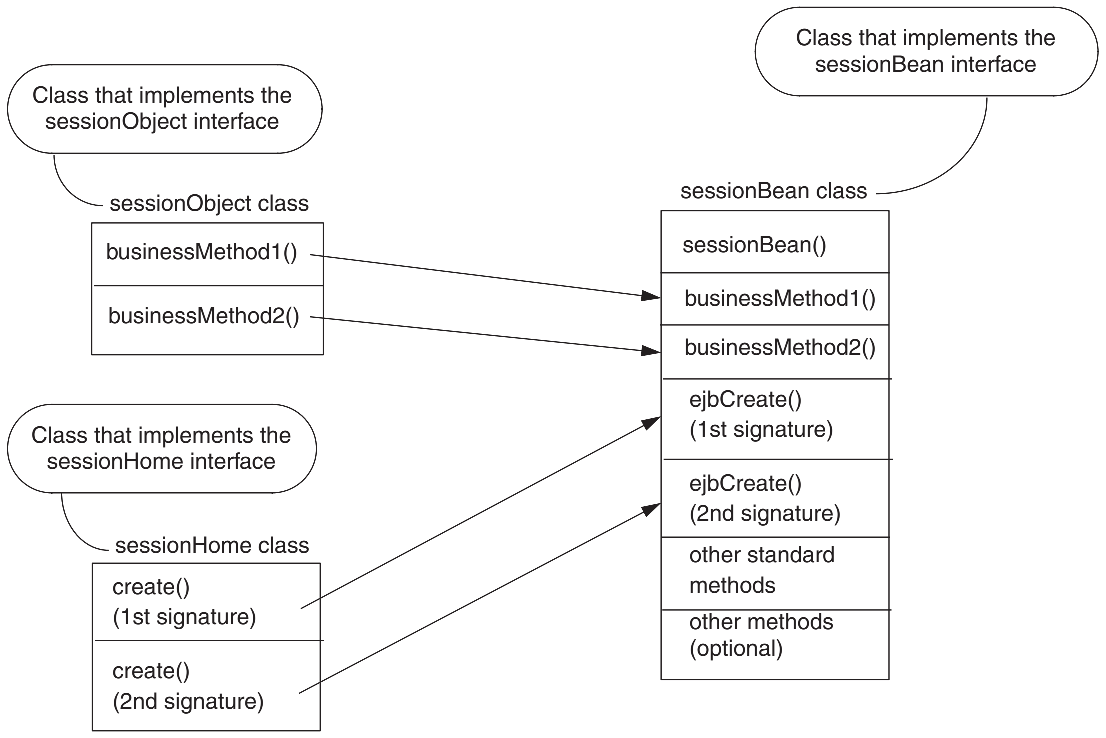


Figure 5. Calling methods of the SessionBean class

The `SessionBean` methods use the EIS-specific API to communicate directly with the EIS. By isolating the EIS-specific API calls to the session bean, neither client components nor the application server need to know this API. Instead, the client component uses calls in the home and remote calls to request EIS services through the session bean.

Deployment descriptor

An enterprise bean is deployed within an EJB container. At deployment of the enterprise bean, the container generates implementations for both the home and remote interfaces of the enterprise bean. The container reads a **deployment descriptor** to obtain the EJB-specific information it needs. This deployment descriptor, called `ejb-jar.xml`, is an XML file that the EJB provider initializes with information about its enterprise bean. This information includes the names of the Java interfaces that define the home and remote interfaces. In this way, the EJB container can build the custom interfaces that the client component needs to access the enterprise bean.

Server Access for EJB support for EJB technology

In support of the Enterprise JavaBeans specification 1.1, IBM provides WebSphere InterChange Server Access for Enterprise JavaBeans (EJB). With this product, you create a session bean called a **WebSphere Access EJB** (often called just an Access EJB). This session bean enables J2EE client components to 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 EJB brings the following benefits to client components in a J2EE application:

- The client component obtains access to IBM WebSphere Business Integration Collaborations services, which provide valuable access to a variety of business processes.

  WebSphere business integration system provides many collaborations through its Exchange. In addition, you can create custom collaborations. By providing connectivity for an enterprise bean to IBM WebSphere Business Integration Collaborations, the bean can essentially execute complex business logic, which could span application and enterprise boundaries.
- The client component can isolate its business logic from the actual EIS applications that contain the necessary data.

  The client component requests execution of a collaboration, which contains the business logic; ICS manages access to the necessary EISs. The client is *not* required to develop any code to access ICS.
- By requesting execution of IBM WebSphere Business Integration Collaborations, client components can send and receive information in any EIS application that ICS manages (called ICS-managed EISs).

  Through collaborations and IBM WebSphere Business Integration Adapters, a client component can have enterprise connectivity to the many EIS applications that ICS manages.
- The client component obtains indirect use of other WebSphere business integration system features, including data transformation and relationship services.

Figure 6 shows the support WebSphere business integration system provides for the middle tier of an N-tier architecture (shown in Figure 2 on page 3).

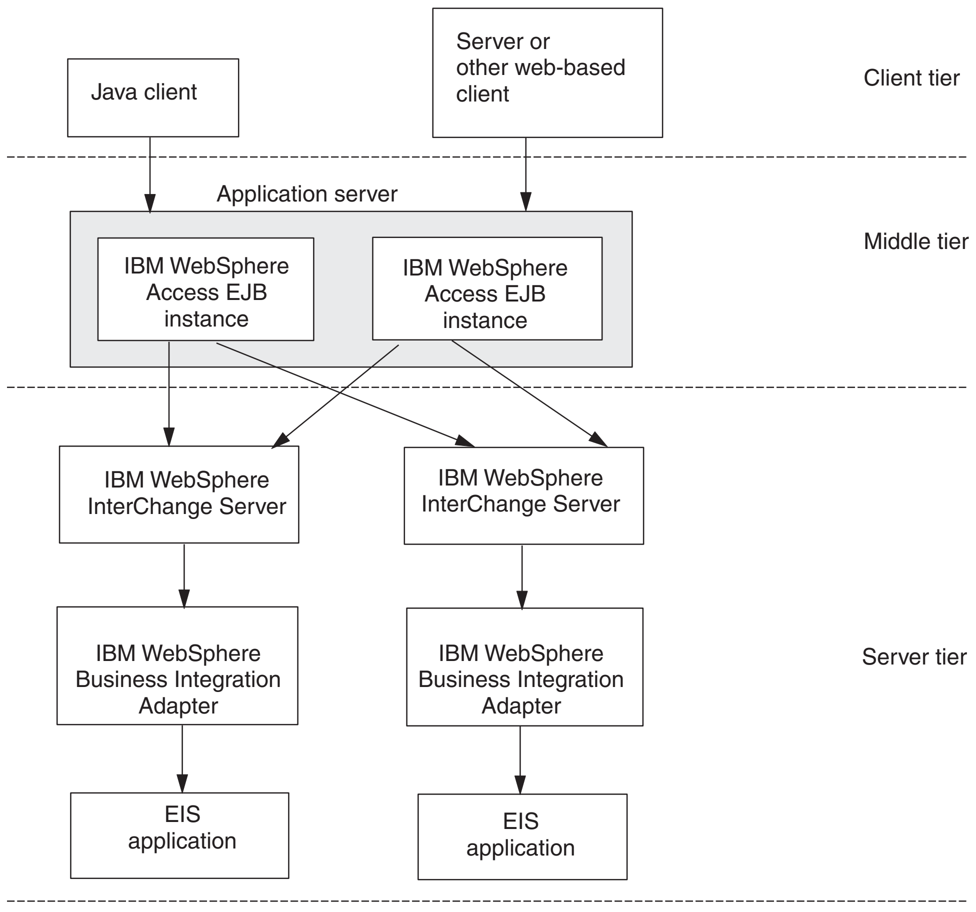

Java client
Server or other web-based client
Client tier
Application server
Middle tier
IBM WebSphere Access EJB instance
IBM WebSphere Access EJB instance
IBM WebSphere InterChange Server
IBM WebSphere InterChange Server
IBM WebSphere Business Integration Adapter
IBM WebSphere Business Integration Adapter
Server tier
EIS application
EIS application

Figure 6. WebSphere business integration system role in N-tier architecture

As Figure 6 shows, WebSphere business integration system provides the following components for middle-tier support:

- WebSphere business integration system-certified application servers
- Instances of WebSphere Access EJBs

Certified application servers

WebSphere business integration system has certified WebSphere Application Server, Version 4.0 Advanced Edition as an application server for use with a WebSphere Access EJB.

Note: A WebSphere Access EJB works with any J2EE application server that supports the Enterprise JavaBeans specification 1.1.

This application server provides basic resource-allocation services to enterprise beans that access EISs. For a WebSphere Access EJB, each application server provides support for the middleware services summarized in Table 3.

Table 3. Middleware services that support a WebSphere Access EJB

Middleware service	WebSphere business integration system support
Connection management	Yes
Security management	Minimum

Table 3. Middleware services that support a WebSphere Access EJB (continued)

Middleware service	WebSphere business integration system support
Transaction management	No However, you can achieve some transaction management through the use of transactional collaborations. For more information, see the *Collaboration Development Guide*.

WebSphere Application Server, Version 4.0 Advanced Edition provides support for the following:

- Deployment tools generate the implementations of the home and remote interfaces when the Access EJB is deployed within the application server.

  For more information on deploying an Access EJB within these application servers, see "Configuring Server Access for EJB" on page 23.
- An EJB container class provides a run-time environment for an Access EJB.

  At run time, an EJB container manages interactions between the WebSphere Access EJB instance and the client component through the home and remote interfaces that Server Access for EJB provides, as Figure 7 shows.

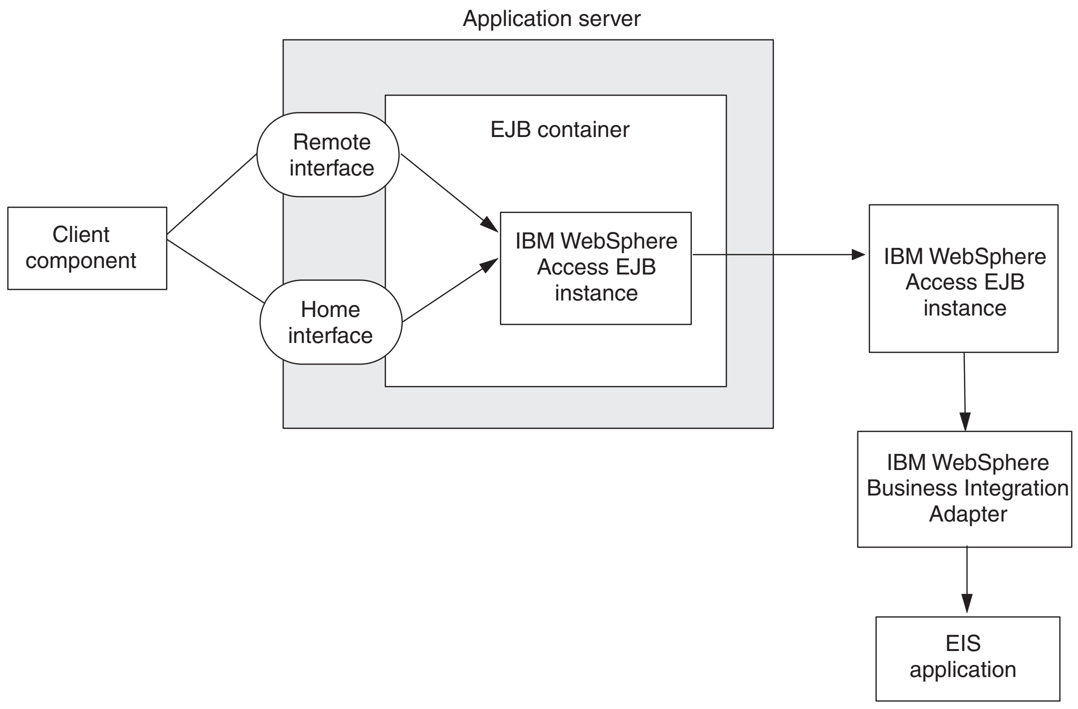

Application server
EJB container
Remote interface
Home interface
Client component
IBM WebSphere Access EJB instance
IBM WebSphere Access EJB instance
IBM WebSphere Business Integration Adapter
EIS application

Figure 7. Accessing an ICS-managed EIS through IBM WebSphere Access EJB

IBM WebSphere Access EJB

A WebSphere Access EJB enables a client 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 system solution initiates execution of a collaboration at the request of the Access EJB. The Access EJB 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 EJBs can send and receive information in the ICS-managed EISs.

To communicate with an ICS-managed EIS, a J2EE client component sends a request to a WebSphere Access EJB, which 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 EJB product includes the following components:

- Implementation of the WebSphere Access EJB

  To provide communication between the client component and the Access EJB, the client component calls either the home or remote interface of the Access EBJ. The client component (in the client tier) interacts with the Access EJB to request execution of a business process (in the form of a collaboration) in an ICS-managed EIS.
- The client-side version of WebSphere InterChange Server Access

  To provide communication between the Access EJB and ICS, each of these components uses WebSphere InterChange Server Access. The Access EJB (in the middle tier) communicates with InterChange Server through InterChange Server Access to send the request for collaboration execution to ICS (in the server tier).

The client component and Access EJB

The WebSphere Access EJB session bean encapsulates the operation of collaboration execution, thereby hiding from the client component the details of initiating and subsequently communicating with InterChange Server. This session bean is stateless; that is, it does *not* support exchange of information between the methods of the session bean. However, because an Access EJB instance is stateless, all instances are equivalent. Therefore, the EJB container can access the instances interchangeably and can delegate the client requests to any available instance.

Server Access for EJB provides the following elements to an application server for the Access EJB's deployment within an EJB container:

- The Java classes to define the interfaces that access the WebSphere Access EJB
  - The session bean's remote interface, `EJBObject`
  - The session bean's home interface, `EJBHome`
- A Java class that implements the `SessionBean` interface for the WebSphere Access EJB
- The `ejb-jar.xml` deployment descriptor with deployment information for the WebSphere Access EJB

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

Home and remote interfaces: A client component communicates with a WebSphere Access EJB through the methods of its home and remote interfaces. Server Access for EJB provides the definitions of these two interfaces, while the deployment tools of the application server generate their implementations (see Figure 4).

Home interface: The **home interface** of the Access EJB provides a J2EE client component with the methods it needs to control the life cycle of the bean. Therefore, Server Access for EJB provides a Java class file that contains the definition of the Access EJB home interface, including the definitions for the following interface methods:

- A single method requests creation of a new Access EJB instance.

The home interface provides the client component with a single `create()` method, which takes no arguments and returns an Access EJB instance.

Note: The signature of `create()` in the home interface's definition matches the signature of its implementation in the Server Access for EJB's implementation of the `SessionBean` interface.

When called from a client component, the `create()` method requests the creation of an Access EJB, which handles communication with an InterChange Server instance. However, it is the responsibility of the EJB container within the application server to handle the actual instances of the Access EJB. The application server checks its pool of available Access EJB instances. If a matching instance exists, it returns to the client a handle to that instance. Otherwise, it initiates creation of a new Access EJB instance and returns its handle to the client. For information, see "Creating a session object" on page 29.

- A `remove()` method requests removal of an existing Access EJB instance.

  Note: The signature of `remove()` in the home interface's definition matches the signature of its implementation in the Server Access for EJB's implementation of the `SessionBean` interface.

  For more information, see "Removing the session object" on page 30.

Remote interface: The **remote interface** of the Access EJB provides a J2EE client component with the methods it needs to execute the EJB's business methods. Therefore, Server Access for EJB provides a Java class file that contains the definition of the Access EJB remote interface. This remote interface provides the `executeCollaborationExtended()` and `executeCollaborationExtendedWithLocale()` methods.

Note: The signature of the `executeCollaborationExtended()` and `executeCollaborationExtendedWithLocale()` methods in the remote interface's definition matches the signature of its implementation in the Server Access for EJB's implementation of the `SessionBean` interface.

For more information, see "Sending data to an ICS-managed EIS" on page 31.

SessionBean class: To implement the Access EJB, Server Access for EJB extends the `SessionBean` interface of the `javax.ejb` package to provide a Java class that implements the `SessionBean` interface. The methods in `SessionBean` use IBM WebSphere InterChange Server Access to communicate directly with InterChange Server. By isolating the Server Access Interface calls to the session bean, neither client components nor the application server need to know this API. Instead, the client component uses calls in the Access EJB home and remote interfaces to request ICS services.

Figure 8 shows the WebSphere business integration system implementation of the EJB Architecture.

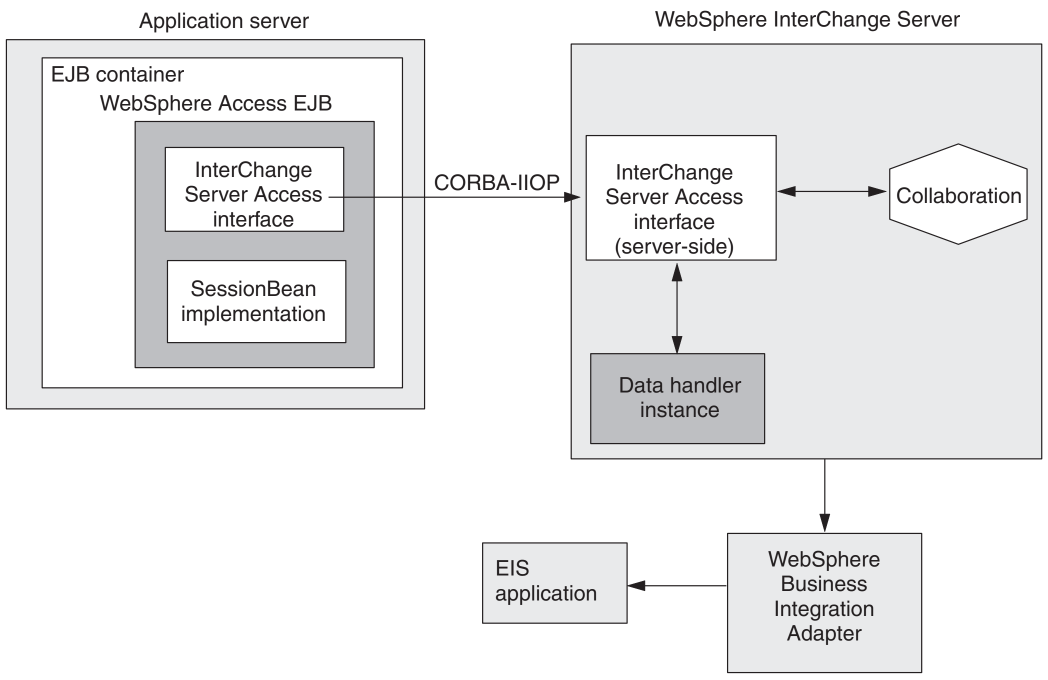

Application server
EJB container
WebSphere Access EJB
InterChange Server Access interface
SessionBean implementation
CORBA-IIOP
WebSphere InterChange Server
InterChange Server Access interface (server-side)
Collaboration
Data handler instance
EIS application
WebSphere Business Integration Adapter

Figure 8. IBM WebSphere Access EJB communication to ICS

This `SessionBean` class contains the implementations of the following methods:

- One `ejbCreate()` method that matches the single `create()` method in the home interface
- Two business methods, `executeCollaborationExtended()` and `executeCollaborationExtendedWithLocale()`, that match the business methods in the remote interface
- Other standard methods of the `SessionBean` interface

Create method of the session bean: server Access for EJB's `SessionBean` class contains the implementation of the single create method of an Access EJB. Within `SessionBean`, this create method is called `ejbCreate()`. The `ejbCreate()` method contains the bean-specific implementation of the home interface's `create()` method; it performs the necessary calls to InterChange Server Access to obtain a connection with InterChange Server. The EJB container includes a call to `ejbCreate()` in its implementation of `create()`.

Note: The signature of `ejbCreate()` in the `SessionBean` class matches the signature of `create()` in the definition of the WebSphere Access EJB home interface. It is `create()` that is surfaced to client components.

For more information, see "Creating a session object" on page 29.

Business method of the session bean: Server Access for EJB's `SessionBean` class contains the implementation of the business methods of the Access EJB, `executeCollaborationExtended()` and `executeCollaborationExtendedWithLocale()`. These business methods enable execution of IBM WebSphere Business Integration Collaborations by client components. They performs the necessary calls to the InterChange Server Access to initiate the execution of a specified collaboration within a specified ICS instance. Upon successful execution of the methods, the Access EJB returns the data in the format that the client has requested.

Note: The signature of the `executeCollaborationExtended()` and `executeCollaborationExtendedWithLocale()` methods in the `SessionBean` class matches the signature in the WebSphere Access EJB remote interface's definition.

For more information, see "Sending data to an ICS-managed EIS" on page 31.

Standard interface methods: Server Access for EJB's `SessionBean` class contains implementations of the other required methods in the `SessionBean` interface. Table 4 summarizes these standard `SessionBean` methods.

Table 4. Other standard methods in SessionBean interface

SessionBean method	Description
`ejbActivate()`	Because the Access EJB is a stateless session bean, this method is not implemented.
`ejbPassivate()`	Because the Access EJB is a stateless session bean, this method is not implemented.
`ejbRemove()`	Contains the bean-specific implementation of the `remove()` method in the home interface; it performs the necessary cleanup activities before the bean instance is removed. The EJB container includes a call to `ejbRemove()` in its implementation of `remove()`.
`setSessionContext()`	Called by the EJB container to associate a bean with a specific session context. A session context is a bean's mechanism for interacting with a container, including obtaining environmental, transactional and security-related information.

Deployment descriptor: Server Access for EJB provides a customized EJB deployment descriptor (`ejb-jar.xml`) for a WebSphere Access EJB to be deployed within the EJB container. At deployment, the EJB container generates implementations for both the home and remote interfaces of the Access EJB using information in this deployment descriptor. The container also obtains other information from this deployment descriptor. This `ejb-jar.xml` deployment descriptor is consistent with the XML DTD for an EJB, as specified in the Enterprise JavaBeans specification 1.1. For more information about the internal structure of the Access EJB deployment descriptor, see "Deploying the WebSphere Access EJB" on page 20.

The Access EJB and InterChange Server

To communicate with an ICS-managed EIS, a J2EE client component sends a request to a WebSphere Access EJB, 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 EJB and ICS, IBM provides WebSphere InterChange Server Access.

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

Figure 9 shows the structure of the communication between an Access EJB and ICS.

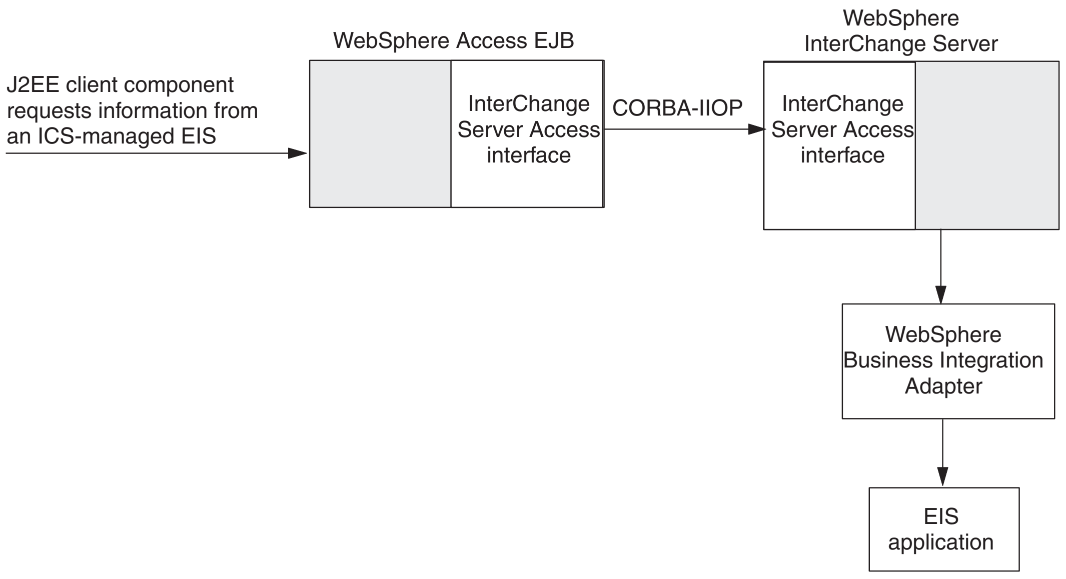

J2EE client component requests information from an ICS-managed EIS
WebSphere Access EJB
InterChange Server Access interface
CORBA-IIOP
WebSphere InterChange Server
InterChange Server Access interface
WebSphere Business Integration Adapter
EIS application

Figure 9. Access to an ICS-managed EIS through InterChange Server Access

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

Overview of client-component development

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

- “Stages in client-component development”
- “Development support for client components” on page 16

Stages in client-component development

To develop a J2EE client component that communicates with an ICS-managed EIS through a WebSphere Access EJB, you code the client-component source file(s) and complete other tasks. The task of creating a client 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 WebSphere InterChange Server Access for EJB, including configuring the development environment for use with an Access EJB. For more information, see “Installing Server Access for EJB” on page 19.
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 client component that uses an Access EJB to execute a collaboration. Implement Java code to create input data, request execution of a collaboration, and access the returned output data.

Figure 10 provides a visual overview of the client-component development process and provides a quick reference to chapters where you can find information on specific topics.

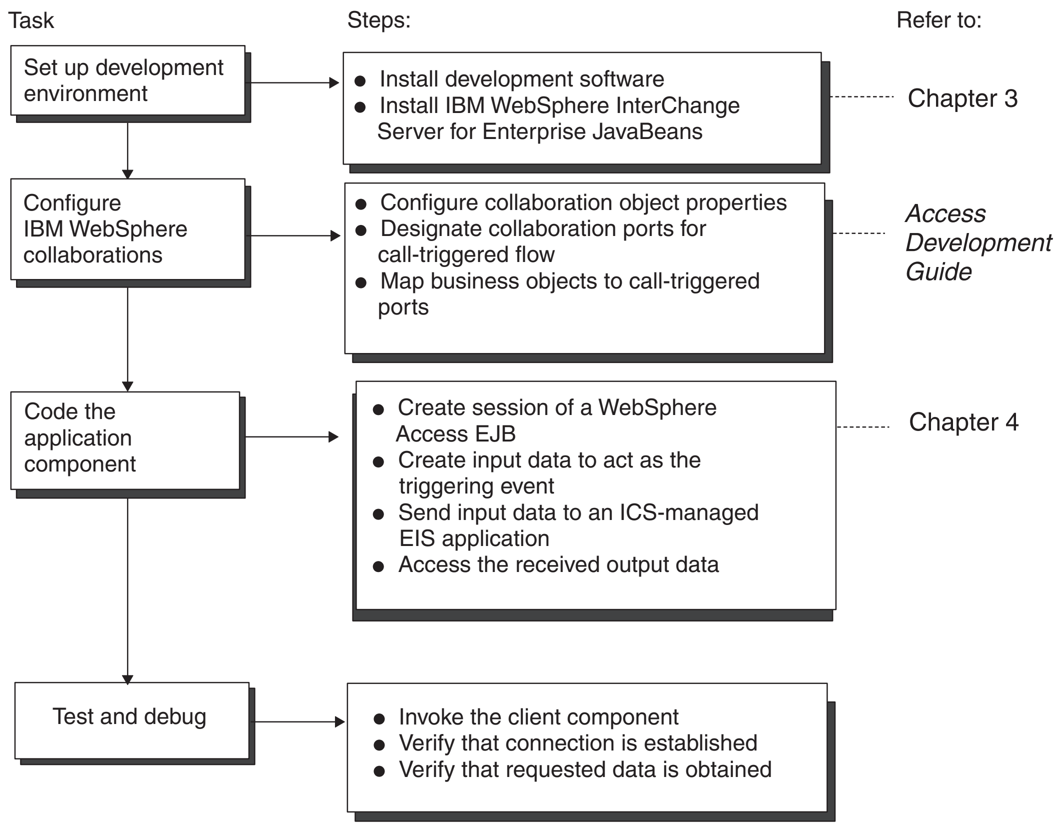

Task
Steps:
Refer to:
Set up development environment
Install development software
Install IBM WebSphere InterChange Server for Enterprise JavaBeans
Chapter 3
Configure IBM WebSphere collaborations
Configure collaboration object properties
Designate collaboration ports for call-triggered flow
Map business objects to call-triggered ports
Access Development Guide
Code the application component
Create session of a WebSphere Access EJB
Create input data to act as the triggering event
Send input data to an ICS-managed EIS application
Access the received output data
Chapter 4
Test and debug
Invoke the client component
Verify that connection is established
Verify that requested data is obtained

Figure 10. Overview of the client-component development task

Development support for client components

To support development of a client component that requires communication with an ICS-managed EIS through an Access EJB, Server Access for EJB provides the features in Table 5.

Table 5. Support for client componets and an Access EJB

Server Access for EJB support	Description
WebSphere Access EJB: `cwsession_ejb-jar.jar`	Provides the client component with ICS-specific implementation of: • The `SessionBean` interface • Definitions of the Access EJB's home and remote interfaces • Deployment descriptor For more information, see "Server Access for EJB support for EJB technology" on page 7.
EJB samples	Includes the following: • Sample client application that issues requests to the sample EJB • Sample servlet that executes business methods of WebSphere Access EJB • Sample EJB

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

```
DevelopmentKits\j2ee\ejb
```

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

The sample provides an EJB that runs within the context of the application server. The sample client application demonstrates how a client component requests the execution of a collaboration through a sample EJB. This sample contains code to perform the following tasks:

- Request the creation of a session of the CwSample EJB.
- Execute the business method `execute()`:
  - Read a test file whose name is established in the `testFile` environment entry of the deployment descriptor.
  - Pass the contents of the test file as input data to the `executeCollaborationExtended()` or `executeCollaborationExtendedWithLocale()` method. Assume that this data is in XML format.
  - Using `executeCollaborationExtended()` or `executeCollaborationExtendedWithLocale()`, execute the `XmlCollab` collaboration within the ICS instance identified by the `icServer` environment entry of the deployment descriptor.

    Note: This `icServer` environment entry is *not* the same entry as the `ICServers` environment entry of WebSphere Access EJB.
- Retrieve business data from an ICS-managed EIS.
- Request the removal of the CwSample EJB session.

Processing locale-dependent data

Server Access for EJB has been internationalized so that it can be localized for a particular locale. A **locale** is the part of a user’s environment that describes conventions specific to the end user’s particular country, language, or territory.

When the WebSphere Access EJB transfers data from a location that uses one character code set to a location that uses a different code set, the system performs character conversion to preserve the meaning of the data. The Java run-time environment within the Java Virtual Machine (JVM) represents data in the Unicode character code set. Unicode contains encodings for characters in most known character code sets (both single-byte and multibyte). Most components in the InterChange Server system are written in Java. Therefore, when data is transferred between most InterChange Server components, there is no need for character conversion.

Chapter 2. Installing and configuring Server Access for Enterprise JavaBeans

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

- "Integration-broker compatibility"
- "Development environment requirements"
- "Installing Server Access for EJB"
- "Configuring Server Access for EJB" on page 23

Integration-broker compatibility

WebSphere InterChange Server Access for EJB is supported on IBM WebSphere Business Integration Adapter Framework version 2.2.0 and later, IBM WebSphere InterChange Server version 4.1.1 and 4.2 and later (if the environment uses ISO Latin-1 data only), WebSphere MQ Integrator version 2.1.0, and WebSphere MQ Integrator Broker, version 2.1.0. See Release Notes for any exceptions.

Development environment requirements

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

- Object Request Broker (ORB) libraries
  - IBM Java Object Request Broker libraries
  - IBM Java ORB 4.5 (or compatible) service
- A Java development environment and Java 2 JDK 1.3.1
- Java run-time libraries 1.3.1
- An application server: WebSphere Application Server, Version 4.0 Advanced Edition

  Note: WebSphere Access EJB works with any J2EE application server that supports the Enterprise JavaBeans specification 1.1. For this release, WebSphere Access EJB has been certified with the WebSphere Application Server listed above.
- WebSphere business integration system software, release 4.0 or later

All of the software listed above *except* the WebSphere business integration system software must reside on the same machine.

Installing Server Access for EJB

As part of the IBM WebSphere InterChange Server installation, InterChange Server Installer installs the files in the directories shown in Table 6 on page 20.. These directory paths are relative to the InterChange Server product directory.

Table 6. Installed file structure for WebSphere Access EJB

Directory	Description
`DevelopmentKits\J2EE\EJB`	Contains the EJB session bean, the WebSphere Access EJB, packaged into a single Java archive (jar) file, called `cwsession_ejb-jar.jar`. This jar file contains the following files: • Java class files for the EJB and its home and remote interfaces • Utility classes to communicate with InterChange Server • Deployment descriptor: `META-INF/ejb-jar.xml` • IBM WebSphere InterChange Server Access stubs
`DevelopmentKits\J2EE\EJB\samples`	Contains source code for EJB samples

Using IBM WebSphere InterChange Server Installer

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

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

  InterChange Server Installer automatically installs the WebSphere Access EJB jar file as part of the ICS installation process. You do *not* need expand the Development Kits for J2EE option of the Installer because the EJB option is automatically selected when you choose the Server option of InterChange Server Installer.
- To install Server Access for EJB *on a different machine* from InterChange Server, run ICS Installer on the machine on which you want to install Server Access for EJB.

  Expand the Development Kits for J2EE option, and select the EJB option.

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

Note: You can install Server Access for EJB on a machine other than the one that contains WebSphere business integration system installation without using ICS Installer. In this case, copy the WebSphere Access EJB jar file (`cwsession_ejb-jar.jar`) into a deployment directory on the machine on which it is to run (the same machine as the application server).

Deploying the WebSphere Access EJB

The Enterprise JavaBeans specification 1.1 requires that an EJB provider define common EJB properties in an XML file named `ejb-jar.xml`. The WebSphere Access EJB jar file contains this deployment descriptor, which provides information that the EJB container needs to deploy WebSphere Access EJB in an application server. The deployment descriptor contains an `<ejb-jar>` element, which is the root element of the deployment descriptor. This element contains information about the WebSphere Access EJB. All general information is represented in string format.

Table 7 on page 21 shows the elements in the deployment descriptor of the WebSphere Access EJB.

Table 7. Contents of the WebSphere Access EJB deployment descriptor

Deployment descriptor element	Description
`<enterprise-beans>`	Marks the beginning of the subsection within the `<ejb-jar>` element that provides the structure of an enterprise bean.
`<session>`	Marks the beginning of the subsection within the `<enterprise-beans>` section that provides the definition of a session bean.
`<ejb-name>`	Local name of the enterprise bean; this name does not have to match the JNDI name that the Deployer assigns to the enterprise bean: `com.crossworlds.access.business. cwsession.CwSession`
`<home>`	The fully qualified name of the Java class that implements the home interface (`EJBHome`) for the WebSphere Access EJB: `com.crossworlds.access.business. cwsession.CwSessionHome`
`<remote>`	The fully qualified name of the Java class that implements the remote interface (`EJBObject`) for the WebSphere Access EJB: `com.crossworlds.access.business. cwsession.CwSession`
`<ejb-class>`	The fully qualified name of the Java class that implements a WebSphere Access EJB (`SessionBean`): `com.crossworlds.access.business. cwsession.CwSessionBean`
`<session-type>`	Because a WebSphere Access EJB is a stateless session bean, this value is set to: `Stateless`
`<transaction-type>`	Because the transaction demarcation for a WebSphere Access EJB is performed by the EJB container, this value is set to: `Container`
`<env-entry>`	These environment entries provide information for each ICS instance that a WebSphere Access EJB can access. For more information, see “Configuring Server Access for EJB” on page 23.
`<assembly-descriptor>`	Marks the beginning of the subsection within the `<ejb-jar>` element that describes how the Enterprise JavaBeans in this deployment descriptor are assembled into a larger application deployment unit.
`<container-transaction>`	Marks the beginning of the subsection within the `<assembly-descriptor>` section that provides declaration information about transaction attributes.
`<method>`	Marks the beginning of the subsection within the `<container-transaction>` section that provides information about the methods in the WebSphere Access EJB remote interface that require transaction attributes.
`<trans-attribute>`	Because a collaboration executes within its own transaction context within ICS, this value is set to: `NotSupported` If a client component calls with a transaction context, the EJB container suspends the association of the transaction context with the current thread before it invokes either the `executeCollaborationExtended()` or `executeCollaborationExtendedWithLocale()` methods.

Note: The `ejb-jar.xml` deployment descriptor is consistent with the XML DTD for an EJB, as specified in the Enterprise JavaBeans specification 1.1.

Figure 11 shows a deployment descriptor of the WebSphere Access EJB, which is contained in the `ejb-jar.xml` file. You do not usually need to edit this file. Most application servers provide tools that provide access to the deployment descriptor.

```
<ejb-jar>
   <enterprise-beans>
      <session>
      <ejb-name>com.crossworlds.access.business.cwsession.CwSession</ejb-name>
      <home>com.crossworlds.access.business.cwsession.CwSessionHome</home>
      <remote>com.crossworlds.access.business.cwsession.CwSession</remote>
      <ejb-class>
         com.crossworlds.access.business.cwsession.CwSessionBean
         </ejb-class>
      <session-type>Stateless</session-type>
      <transaction-type>Container</transaction-type>
      <env-entry>
         <description>
            The list of interchange servers which this bean would access. It would
            be of the format ICServer1, ICServer2
            </description>
         <env-entry-name>ICServers</env-entry-name>
         <env-entry-type>java.lang.String </env-entry-type>
         <env-entry-value>CrossWorlds,dexter</env-entry-value>
         </env-entry>
      <env-entry>
         <description>Trace Level for the bean </description>
         <env-entry-name>traceLevel</env-entry-name>
         <env-entry-type>java.lang.Integer </env-entry-type>
         <env-entry-value>5</env-entry-value>
         </env-entry>

<!--  COMMENT: Now we will have entries for each of the InterChange Server instances listed above -->
   <!-- The First ICS instance: CrossWorlds -->
      <env-entry>
         <env-entry-name>CrossWorlds/userName</env-entry-name>
         <env-entry-type>java.lang.String</env-entry-type>
         <env-entry-value>admin</env-entry-value>
         </env-entry>
      <env-entry>
         <env-entry-name>CrossWorlds/passWord </env-entry-name>
         <env-entry-type>java.lang.String</env-entry-type>
         <env-entry-value>null</env-entry-value>
         </env-entry>
      <env-entry>
         <env-entry-name>CrossWorlds/iorLocation </env-entry-name>
         <env-entry-type>java.lang.String</env-entry-type>
         <env-entry-value>
            c:/CrossWorlds/CrossWorldsInterchangeServer.ior</env-entry-value>
            </env-entry>
      <env-entry>
         <env-entry-name>CrossWorlds/numAccessSessions </env-entry-name>
         <env-entry-type>java.lang.Integer</env-entry-type>
         <env-entry-value>10 </env-entry-value>
         </env-entry>
```

Figure 11. WebSphere Access EJB deployment descriptor (Part 1 of 2)

```
   <!-- The Second ICS instance: dexter -->
      <env-entry>
         <env-entry-name>dexter/userName</env-entry-name>
         <env-entry-type>java.lang.String</env-entry-type>
         <env-entry-value>admin</env-entry-value>
         </env-entry>
      <env-entry>
         <env-entry-name>dexter/passWord </env-entry-name>
         <env-entry-type>java.lang.String</env-entry-type>
         <env-entry-value>null</env-entry-value>
         </env-entry>
      <env-entry>
         <env-entry-name>dexter/iorLocation </env-entry-name>
         <env-entry-type>java.lang.String</env-entry-type>
         <env-entry-value>
            D:/CrossWorlds2k/dexterInterchangeServer.ior
            </env-entry-value>
         </env-entry>
      <env-entry>
         <env-entry-name>dexter/numAccessSessions </env-entry-name>
         <env-entry-type>java.lang.Integer</env-entry-type>
         <env-entry-value>10 </env-entry-value>
      </env-entry>
   </session>
   </enterprise-beans>

   <assembly-descriptor>
   <container-transaction>
         <method>
            <ejb-name>
               com.crossworlds.access.business.cwsession.CwSession
               </ejb-name>
            <method-intf>Remote</method-intf>
            <method-name>*</method-name>
            </method>
         <trans-attribute>NotSupported</trans-attribute>
      </container-transaction>
   </assembly-descriptor>
</ejb-jar>
```

Figure 11. WebSphere Access EJB deployment descriptor (Part 2 of 2)

Important: The WebSphere Access EJB deployment descriptor in Table 11 has been formatted differently from the actual `ejb-jar.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. For the most accurate version of this file, use the appropriate application-server tool to view the actual `ejb-jar.xml` file on your system.

For information on how to customize the `<env-entry>` elements, see "Setting environment entries" on page 24.

Configuring Server Access for EJB

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

- Making the Server Access for EJB ITLM agent enabled
- Setting environment entries
- Synchronizing the `.ior` file
- Configuring the application server

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

Making the Server Access for EJB ITLM agent enabled

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

```
CC000976J040300.sys
```

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

Setting environment entries

When your WebSphere Access EJB is deployed within an application server, the EJB container sets the environment for the bean. The **environment** of an enterprise bean provides the mechanism to customize the business logic of the bean during deployment or assembly. This environment comprises **environment entries**, each of which specifies of piece of environment information.

The deployment descriptor provides `<env-entry>` elements to specify each environment entry. By reading the deployment descriptor, the EJB container is able to initialize the enterprise bean's environment. Table 8 shows the information that each `<env-entry>` element provides.

Table 8. Environment entry information

Subelement of <env-entry>	Description
`<description>`	An optional description of the environment entry
`<env-entry-name>`	The name of the environment entry
`<env-entry-type>`	The Java type of the environment entry; this type is what is expected to be returned from the JNDI `lookup()` method
`<env-entry-value>`	The value to assign the environment entry

To configure the environment of your Access EJB, you specify environment entries in its deployment descriptor. Table 9 shows the environment entries that Server Access for EJB supports.

Table 9. Server Access for EJB environment entries

Property name	Description
`ICServers`	A comma-separated list of the names of the ICS instances that this enterprise bean can access.
`traceLevel`	The trace level that determines whether debugging information is generated for the WebSphere Access EJB. For more information, see "Tracing" on page 37.
`logFile`	Name of the log file for the WebSphere Access EJB. This name must be the file's absolute path name. For more information, see "Logging" on page 36.
`<`*`ICSName`*`>/userName`	Account name to use when connecting to the *ICSName* instance of ICS. The *only* user name currently supported is `admin`.
`<`*`ICSName`*`>/passWord`	Password for the `admin` user.

Table 9. Server Access for EJB environment entries (continued)

Property name	Description
`<`*`ICSName`*`>/iorLocation`	Name of the Interoperable Object Reference (.ior) file for the `ICSName` instance of ICS. This name must be the file's absolute path name.
`<`*`ICSName`*`>/numAccessSessions`	Maximum number of simultaneous connections allowed to the `ICSName` instance of ICS. The default value is one (1).

Note: Setting the environment entries in Table 9 on page 24 is the *only* way to configure a WebSphere Access EJB. You *cannot* configure it through System Manager.

For each ICS instance that your enterprise bean accesses, you must take the following steps in the deployment descriptor:

1. List the name of the ICS instance in the `ICServers` environment entry; if an ICS name already exists in this entry, separate ICS names with a comma (,).

   If this environment entry is missing or if it does not contain ICS instances that are described in step 2, WebSphere Access EJB throws a `CreateException` exception.
2. Provide the ICS-specific information for the ICS instance:
   - Name of user account (`admin`) in the `username` environment entry
   - Name of the password for the user account in the `passWord` environment entry
   - Absolute path name for the ICS-specific `.ior` file in the `iorLocation` environment entry
   - Maximum number of simultaneous connections in the `numAccessSessions` environment entry

   Each of the above entries has the string "*`ICSName/`*" prepended, where *`ICSName`* is the name of the ICS instance. If any of this information is missing, the bean does *not* create any connections to that ICS instance.

The environment entries are specified *only once* in the deployment descriptor. However, all instances of the enterprise bean in the same home share the same environment.

Handling the .ior file

At run time, Server Access for EJB 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 EJB 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 EJB). To obtain the location of the ORB server, an Access EJB 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 EJB can then use this file to locate the ORB server and, in turn, to communicate with its ICS instance.

Therefore, for an Access EJB 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 EJB resides is able to locate the `.ior` file for its InterChange Server instance.
3. Synchronize the `.ior` file between the WebSphere Access EJB and InterChange Server.

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

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 EJB cannot depend on the `.ior` file to locate the ORB server. Therefore, an Access EJB 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 12 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 12. 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 12 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 EJB 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 EJB 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-EJB machine can access the `.ior` file:

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

Synchronizing the .ior file

The WebSphere Access EJB obtains the name of the InterChange Server instance with which it communicates from the ICS-specific Interoperable Object Reference (`.ior`) file. The enterprise bean obtains the name of this `.ior` file from the `iorLocation` environment entry. Therefore, to synchronize the Access EJB and InterChange Server, you must set this `iorLocation` environment entry to contain the absolute path name for the ICS-specific `.ior` file for the machine on which Server Access for EJB resides.

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

Suppose that the Server Access for EJB deployment directory is:

ProductDir`\DevelopmentKits\J2EE\EJB`

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

ProductDir`\DevelopmentKits\J2EE\EJB\dexterinterchangeServer.ior`

Configuring the application server

This section provides a summary of how to configure the WebSphere Application Server 4.0 Advanced Edition, which is the application server that is certified for use with Server Access for EJB. For more information, refer to the `Readme.txt` file in the subdirectory of the Server Access for EJB samples for the WebSphere Application Server Advanced Edition 4.0:

ProductDir`\DevelopmentKits\J2EE\EJB\samples`

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

- "Modifying environment entries" on page 28
- "Deploying the enterprise bean" on page 28
- "Making ORB libraries available" on page 28

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

Modifying environment entries

To update configuration properties, you change their values in the deployment descriptor for Server Access for EJB, `ejb-jar.xml`. This deployment descriptor is part of the `cwsession_ejb-jar.xml` file. It is recommended that you use the administration console of your application server after the enterprise bean 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 environment entries, see "Configuring Server Access for EJB" on page 23.

Deploying the enterprise bean

To deploy an enterprise bean for use with WebSphere Application Server, you must install the external .jar file and deploy an enterprise bean. For more information on these steps, see the documentation for WebSphere Application Server.

Making ORB libraries available

At run time, Server Access for EJB 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 runtime. 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 Enterprise JavaBeans

This chapter provides information on how to use a WebSphere Access EJB from a client component to obtain information from an EIS application managed by InterChange Server (ICS). It provides information on how to use this Access EJB to initiate execution of an IBM WebSphere Business Integration Collaboration.

This chapter contains the following sections:
- "Managing the session object"
- "Sending data to an ICS-managed EIS" on page 31
- "Obtaining session metadata" on page 35
- "Handling exceptions" on page 36

Managing the session object

An instance of WebSphere Access EJB handles the connection to ICS as well as the initiation of a collaboration within ICS. A client component can manage this EJB instance through the methods of the Access EJB home interface, shown in Table 10.

Table 10. Methods in the EJBHome interface

EJBHome method	Description
`create()`	Creates a session object for WebSphere Access EJB, which obtains an instance of WebSphere Access EJB
`getEJBMetaData()`	Obtains an instance of the `EJBMetaData` class, with which a client component can obtain metadata. For more information, see "Obtaining session metadata" on page 35.
`remove()`	Removes a session object for WebSphere Access EJB

The interfaces in Table 10 provide the following session-management tasks:
- "Creating a session object"
- "Removing the session object" on page 30

Creating a session object

To create a session object for a WebSphere Access EJB, the client component must obtain an `EJBObject` instance, which represents an instance of Access EJB. The client component is responsible for requesting that an Access EJB instance be created or removed. However, the EJB container within the application server determines when such activities actually occur. The container creates a pool of Access EJB instances. Because Access EJB is a stateless session bean, all its instances are equivalent. Therefore, the container can delegate the client request to any available instance.

To acquire an instance of WebSphere Access EJB, the client component takes the following steps:
1. Look up the home interface for a WebSphere Access EJB from the Java Naming and Directory Interface (JNDI).

When the Access EJB is deployed, the EJB container registers its home interface in the JNDI. The JNDI name for the Access EJB is defined within the application server. This JNDI lookup returns a reference to an `EJBHome` instance.

2. Perform type-narrowing of the returned JNDI reference returned from step 1 on page 29 with the `narrow()` method of the Java Remote Method Invocation (RMI).

   The Enterprise JavaBeans specification 1.1 recommends the following with respect to type-narrowing:

   "A client component that is intended to be interoperable with all compliant EJB Container implementations must use the `javax.rmi.PortableRemoteObject.narrow(...)` method to perform type-narrowing of the client-side representations of the home and remote interfaces."
3. Create a session object for the Access EJB with the `EJBHome.create()` method.

   The home interface provides a single `create()` method to create an instance of a Access EJB. This method takes no arguments and returns the `EJBObject` instance for Access EJB.

The following code fragment shows a client component creating a session object for an Access EJB:

```
InitialContext ic = getInitialContext();

// Look up the home interface for WebSphere Access EJB
Object ref = context.lookup(
   "com/crossworlds/access/business/cwsample/CwSample");
CwSampleHome cwHome = (CwSampleHome)
   javax.rmi.PortableRemoteObject.narrow(ref, CwSampleHome.class);

// Create a session object
CwSample sample = cwHome.create();

// code to request execution of the collaboration goes here
.....

// Remove the session
cwHome.remove();
```

When the client component requests a new session object, the application server checks its pool for an available Access EJB instance. If a session object already exists in one of the application server's pools that can satisfy the session request, the `create()` method returns a handle to that session. Otherwise, the method requests the creation of a new session object of an Access EJB. If one does *not* exist, the EJB container obtains a new one with the following steps:

1. Call the `newInstance()` method to create the actual instance of an Access EJB.
2. Call the `setSessionContext()` method to pass the context object to the Access EJB instance.
3. Call the `ejbCreate()` method that corresponds to the `create()` method that the client component specified.

Note: If the EJB container cannot create a new Access EJB instance, it throws an appropriate `RemoteException` exception to notify the client component.

Removing the session object

The EJB package provides two ways in which the client component can remove a session object, as follows:

- In the remote interface: `EJBObject.remove()`

- In the home interface: `EJBHome.remove()`

Important: Because WebSphere Access EJB is a session bean, you *cannot* invoke the `remove(Object` *`primaryKey`*`)` form of the `EJBHome.remove()` method. An attempt to do so results in the generation of a `RemoveException` exception.

When the client component is finished using an Access EJB instance, it should explicitly remove it. However, a call to one of these `remove()` method does *not* necessarily remove the actual Access EJB instance; it just notifies the EJB container that the instance can be removed if no other client components are currently using it.

Sending data to an ICS-managed EIS

WebSphere InterChange Server Access for EJB provides support for a client component to send data to an ICS-managed EIS application through either the `executeCollaborationExtended()` or `executeCollaborationExtendedWithLocale()` method. The client component prepares input data, which represents the triggering event of the collaboration. It then requests execution of the collaboration with either the `executeCollaborationExtended()` or `executeCollaborationExtendedWithLocale()` method of the enterprise bean's remote (`EJBObject`) interface. When a WebSphere Access EJB executes the method, it uses calls to the WebSphere InterChange Server Access Interface to communicate with InterChange Server (see "Business method of the session bean" on page 13). Therefore, the method initiates a call-triggered flow when it submits the triggering event to a collaboration.

The syntax of the `executeCollaborationExtended()` method in the remote interface is as follows:

```
public String executeCollaborationExtended(
       String ICSName,
       String collabName,
       String portName,
       String serializedData,
       String mimeType,
       String verb)
    throws CollaborationExecutionException, RemoteException;
```

The syntax of the `executeCollaborationExtendedWithLocale()` method in the remote interface is as follows:

```
public String executeCollaborationExtended(
       String ICSName,
       String collabName,
       String portName,
       String serializedData,
       String mimeType,
       String verb       String locale)
   throws CollaborationExecutionException, RemoteException;
```

Note: These method signatures match the implementation of the corresponding methods in the Server Access for EJB's implementation of the `SessionBean` interface.

Table 11 on page 32 provides a summary of the parameters in the `executeCollaborationExtended()` and `executeCollaborationExtendedWithLocale()` methods.

Table 11. Parameters of the remote interface methods

Parameter name	Description	For more information
ICSName	Name of the ICS instance that contains the collaboration	"Connecting to the ICS instance" on page 32
collabName	Name of the collaboration to execute	"Executing the collaboration" on page 33
portName	Name of the collaboration port that receives the `serializedData` data	"Executing the collaboration" on page 33
serializedData	Input data, which is interpreted as the triggering event of the collaboration	"Converting the serialized input data" on page 34
mimeType	MIME type of the `serializedData` data	"Converting the serialized input data" on page 34
verb	WebSphere business integration system verb to send with the triggering event	"Executing the collaboration" on page 33
locale	The client locale with which to execute the collaboration.	"Processing locale-dependent data" on page 17

Keep the following points in mind when requesting execution of a collaboration through a WebSphere Access EJB:

- The `executeCollaborationExtended()` and `executeCollaborationExtendedWithLocale()` methods provide *only* the ability to initiate execution of a requested collaboration.

  Server Access for EJB does *not* provide the ability to manipulate individual attributes of a business object or to create a specific business object. Actual manipulation and creation of a business object is delegated to InterChange Server.
- Because this initiated flow is a call-triggered flow, the event is processed synchronously and is *not* persistent within the WebSphere business integration system.

  If ICS unexpectedly terminates during execution of the call-triggered flow, the client component is *not* notified of the failure through an exception. However, the enterprise bean is informed whether the request was processed *before* the termination. To recover:
  - For retrieval requests, the client component can reissue the retrieval request.
  - For requests that modify data in an ICS-managed EIS, the client component must determine whether the request was completed once ICS has been restarted.

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

Execution of the `executeCollaborationExtended()` method involves the following steps:

- "Connecting to the ICS instance"
- "Executing the collaboration" on page 33
- "Converting the serialized input data" on page 34
- "Receiving the output data" on page 35
- "Closing the connection" on page 35

Connecting to the ICS instance

The first step in initiating a call-triggered flow is to obtain a connection with an instance of InterChange Server. To identify the ICS instance that the WebSphere Access EJB is to use, the `executeCollaborationExtended()` method specifies the ICS name as its first argument. This ICS instance contains the collaboration that the client component needs to execute. This name must correspond to one of the ICS definitions in the Server Access for EJB environment, which is established when the WebSphere Access EJB was deployed. The deployment descriptor provides the valid ICS definitions. For more information, see "Setting environment entries" on page 24.

The following call to `executeCollaborationExtended()` establishes a connection to an ICS instance named `dexter`:

```
returnedData = cwObject.executeCollaboration("dexter", ....);
```

The deployment descriptor in Table 9 on page 24 shows the ICS definition for an ICS instance named `dexter`.

Note: A WebSphere Access EJB establishes connection pools that last for the lifetime of the application server. Therefore, the `executeCollaborationExtended( )` method does not necessarily establish a new connection each time it executes. If a collaboration pool contains a connection to the requested ICS instance, `executeCollaborationExtended( )` obtains this connection. For more information, see "Closing the connection" on page 35.

Executing the collaboration

The second step in initiating a call-triggered flow is to execute the collaboration. To identify the collaboration to execute, use the `executeCollaborationExtended()` method. Its arguments provide the Access EJB with information it needs to identify the collaboration to execution. Several of this method's arguments specify the actual collaboration name and port, as follows:

- The second argument specifies the name of the collaboration to initiate.
- The third argument specifies the port of the collaboration that receives the incoming data.
- The last (sixth) argument specifies the WebSphere business integration system verb for the collaboration.

Note: The collaboration and port that you specify must be configured for a 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*.

Once a connection to an ICS instance has been obtained, the `executeCollaborationExtended()` or `executeCollaborationExtendedWithLocale()` method initiates execution of the specified collaboration.

Consider the call to the `executeCollaborationExtended()` method in Figure 13 on page 34. This call provides information about which collaboration to execute in its second, third, and last arguments (in bold). The call performs the following tasks:

- Obtain a connection to an ICS instance named `dexter`.
- Initiate execution of a collaboration named `Customer`.
- Send the input data to the `NewCust` port in the `Customer` collaboration.
- Request a WebSphere business integration system verb of Create.

```
returnedData = cwObject.executeCollaborationExtended(
   "dexter",
   "Customer",
   "NewCust",
   inputData,
   "text/xml"
   "Create");
```

Figure 13. A sample call to executeCollaborationExtended()

Converting the serialized input data

The third step in initiating a call-triggered flow is to send serialized data to represent the triggering event for the collaboration. To identify the serialized data, the `executeCollaborationExtended()` method provides the serialized data and information on the format of this data as arguments, as follows:

- The fourth argument specifies the actual serialized input data.

  Pass this input data as a Java `String`.
- The fifth argument specifies the MIME type of the input data.

  The MIME type specifies the format of the serialized data. You can specify only MIME types that are supported by the ICS instance you request.
- The seventh argument—`locale`, which is only present in the `executeCollaborationExtendedWithLocale()` version of the method—specifies the locale for the input data.

  The MIME type specifies the format of the serialized data. You can specify only MIME types that are supported by the ICS instance you request.

As part of the call-triggered flow, InterChange Server converts the serialized input data into a WebSphere business integration system business object, which is the actual triggering event for the collaboration. To perform this data conversion, ICS uses a **data handler**, which converts between a business object and a particular serialized format (such as XML, name-value pairs, or EDI). Server Access within ICS uses the MIME type to determine which data handler to use. To obtain the triggering event, the data handler converts the serialized input data, which is in the format specified by its associated MIME type, into a business object.

Note: For more information on MIME types and their role in the selection of a data handler, see the *Data Handler Guide*.

In the call to the `executeCollaborationExtended()` method in Figure 13, the fourth and fifth arguments provide the input data and its MIME type. These arguments provide the information that ICS needs to perform the following tasks:

- Choose the XML data handler: MIME type of `text/xml`.
- Use the XML data handler to create a business object from the XML data: data is in the `inputData` variable is XML data.
- Send the resulting business object as the triggering event for the `Customer` collaboration.

Once the collaboration receives its triggering event, it begins execution.

Receiving the output data

The purpose of a client component requesting execution of a collaboration is to obtain information from an ICS-managed EIS. Execution of the collaboration results in the collaboration returning a business object, which contains data from an ICS-managed EIS. As the final step, InterChange Server converts this business object into serialized output data. To perform this data conversion, ICS uses the same data handler that it used to convert the serialized input data into the triggering event. It selects this data handler based on the MIME type that the `executeCollaborationExtended()` method provides as its fifth argument. This data handler converts the returned collaboration business object into the serialized output data, in the format specified by this MIME type.

The client component receives the output data as the return value of the `executeCollaborationExtended()` method. This return value is a serialized `String` value, in the same format as the serialized input data that the client component passed in to `executeCollaborationExtended()`. In the call to the `executeCollaborationExtended()` method in Figure 13, the client component receives its serialized output data in the `returnedData` variable. This variable contains the requested EIS information in XML format.

If any errors occur during collaboration execution, the WebSphere Access EJB sends them to the client component as appropriate. The types of errors that the session bean might encounter include:

- Metadata errors, such as the absence of a business object definition, or a mismatch in the business objects specified
- Data-handler errors, such as the absence of a data handler, or a data conversion error

For more information, see “Exceptions” on page 37.

Closing the connection

A WebSphere Access EJB is responsible for managing the opening and closing of ICS connections. As a performance enhancement, the session bean does *not* automatically close an ICS connection when the collaboration completes. Instead, it keeps a pool of connections, one for each ICS instance defined in the session bean’s environment. These ICS instances are listed in the `ICServers` environment entry of the bean’s deployment descriptor. The Access EJB closes a connection *only* if a communications error occurs; otherwise, these connections are valid for the lifetime of the application server.

Obtaining session metadata

The `EJBMetaData` interface provides information about the enterprise bean. Table 12 on page 36 shows the methods that the `EJBMetaData` interface provides for obtaining metadata for the session bean. A client component can use this metadata to dynamically invoke an enterprise bean.

Table 12. Methods in the EJBMetaData interface

EJBMetaData method	Description
`getEJBHome()`	Obtains the home interface of the WebSphere Access EJB
`getHomeInterfaceClass()`	Obtains the class object for the home interface of the WebSphere Access EJB
`getPrimaryKeyClass()`	*Because a WebSphere Access EJB is a session bean, this method is not implemented.*
`getRemoteInterfaceClass()`	Obtains the class object for the remote interface of the WebSphere Access EJB
`isSession()`	Determines whether the current enterprise bean is a session bean. This method always returns `true` for a WebSphere Access EJB.
`isStatelessSession()`	Determines whether the current session bean is a stateless session bean. This method always returns `true` for a WebSphere Access EJB.

Important: Support for the `EJBMetaData` interface is dependent on the EJB container. The containers for the application servers that are WebSphere business integration system-certified support this metadata interface.

To obtain EJB metadata, the client component takes the following steps:

1. Use the `EJBHome.getEJBMetaData()` method to obtain an `EJBMetaData` object.
2. Use the appropriate `EJBMetaData` method (see Table 12) to obtain the metadata from this object.

Handling exceptions

A WebSphere Access EJB provides the following features for handling exceptions:

- "Logging and tracing"
- "Exceptions" on page 37

Logging and tracing

Debugging an enterprise bean is largely dependent on the application server in which the enterprise bean is deployed. A WebSphere Access EJB handles logging and tracing as described in the following sections.

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

Logging

The application server usually handles in its own server-specific manner all log messages that an enterprise bean generates. Usually, you can configure this structure through a console or an application-server deployment descriptor. The WebSphere Access EJB deployment descriptor provides the `logFile` environment entry to define the location of the log file that this enterprise bean uses.

A WebSphere Access EJB directs log messages based the setting of this environment entry:

1. The enterprise bean looks for the `logFile` environment entry and logs all error and trace messages to it.

   Important: The default deployment descriptor that WebSphere business integration system provides does *not* include the `logFile` environment entry because some application servers automatically reroute all log and trace messages to their own application-server log. If you want to specify a separate log file for an instance of a WebSphere Access EJB, you must add the `logFile` environment entry *before* you deploy the Access EJB.

2. If no `logFile` environment entry has been specified in the deployment descriptor, the WebSphere Access EJB logs all error and trace messages to standard output (STDOUT).

Tracing

As with logging, the application server usually handles in its own server-specific manner all trace messages that an enterprise bean generates. There is no concept of a trace file for a WebSphere Access EJB. The application server logs all trace messages that the enterprise bean generates in its own server-specific file. The WebSphere Access EJB deployment descriptor provides the `traceLevel` environment entry to define the amount of trace information desired, as follows:

- To turn WebSphere Access EJB tracing *on*, set `traceLevel` to a value >= 0.
- To turn WebSphere Access EJB tracing *off*, you can take either of the following actions:
  - Set `traceLevel` to a value less than zero.
  - Remove `traceLevel` from the deployment descriptor.

Exceptions

A WebSphere Access EJB provides support for exception handling. Table 13 shows the types of exceptions that WebSphere Access EJB handles.

Table 13. Types of EJB exceptions

Kind of exception	Description	Handled by
Application exception	An exception that is thrown by a method of the WebSphere Access EJB home or remote interface An application exception does *not* result in the removal of the WebSphere Access EJB instance.	Client component
System exception	An exception not meant to be surfaced to the client component	Application server

Table 14 lists the application exceptions that a WebSphere Access EJB can generate. The client component should check for these application exceptions.

Table 14. Application exceptions

Application exception	Description
`CreateException`	Thrown by a WebSphere Access EJB to indicate an application exception has occurred in a `create()` method.
`RemoveException`	Thrown by a WebSphere Access EJB to indicate an application exception has occurred in a `remove()` method.
`CollaborationExecutionException`	Thrown by a WebSphere Access EJB to indicate that the collaboration execution has encountered an error, such as: • data format conversions • an attempt to execute the collaboration when the session bean is not configured to communicate with the specific ICS instance

Table 14. Application exceptions (continued)

Application exception	Description
`RemoteException`	Thrown by a method of the remote and home interfaces of a WebSphere Access EJB to indicate that it has encountered an error in the way that the method was invoked. Because the remote and home interfaces of a WebSphere Access EJB are Remote Method Invocation (RMI) interfaces, all their methods throw the `RemoteException` exception or one of its subclasses. The application server or EJB container might throw a `RemoteException` subclass to provide additional information.

The exceptions in Table 13 on page 37 are derived from the `Exception` class. A `CollaborationExecutionException` provides the following information:

- A string that describes the error

  This string is a standard Java exception message and is available through the `getMessage()` method.
- A status value that indicates how far into the collaboration execution the error occurred:
  - Failure before executing the collaboration
  - Failure during execution of the collaboration
  - Failure during the communication of the result

Index

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

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 WebSphere InterChange Server v4.3.0