IBM WebSphere Business Integration Adapters

Adapter for EJB User Guide

Version 1.0.x

IBM WebSphere Business Integration Adapters

Adapter for EJB User Guide

Version 1.0.x

Note!
Before using this information and the product it supports, read the information in "Notices" on page 83.

20February2004

This edition of this document applies to IBM WebSphere Business Integration Adapter for EJB, version 1.0.x, and to all subsequent releases and modifications until otherwise indicated in new editions.

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

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

© Copyright International Business Machines Corporation 2003, 2004. All rights reserved.
US Government Users Restricted Rights – Use, duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp.

Contents

© Copyright IBM Corp. 2003, 2004

About this document

The IBM(R) WebSphere(R) Business Integration Adapter portfolio supplies integration connectivity for leading e-business technologies, enterprise applications, legacy, and mainframe systems. The product set includes tools and templates for customizing, creating, and managing components for business process integration.

This document describes installation, configuration, business object development, and troubleshooting for the IBM WebSphere Business Integration Adapter for EJB.

Audience

This document is for consultants, developers, and system administrators who support and manage the WebSphere business integration system at customer sites.

Prerequisites for this document

Users of this document should be familiar with the WebSphere business integration system, with business object and collaboration development, and with the EJB technology.

Related documents

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

You can install related documentation from the following sites:

- For general adapter information; for using adapters with WebSphere message brokers (WebSphere MQ Integrator, WebSphere MQ Integrator Broker, WebSphere Business Integration Message Broker); and for using adapters with WebSphere Application Server:
  - http://www.ibm.com/websphere/integration/wbiadapters/infocenter
- For using adapters with InterChange Server:
  - http://www.ibm.com/websphere/integration/wicserver/infocenter
  - http://www.ibm.com/websphere/integration/wbicollaborations/infocenter
- For more information about message brokers (WebSphere MQ Integrator Broker, WebSphere MQ Integrator, and WebSphere Business Integration Message Broker):
  - http://www.ibm.com/software/integration/mqfamily/library/manualsa/
- For more information about WebSphere Application Server:
  - http://www.ibm.com/software/webservers/appserv/library.html

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

© Copyright IBM Corp. 2003, 2004

Typographic conventions

This document uses the following conventions:

`courier font`	Indicates a literal value, such as a command name, filename, 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 outline	A blue outline, which is visible only when you view the manual online, indicates a cross-reference hyperlink. Click inside the outline 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 product pathnames are relative to the directory where the connector is installed on your system.
`%text%` and *`$text`*	Text within percent (%) signs indicates the value of the Windows™ `text` system variable or user variable. The equivalent notation in a UNIX environment is *`$text`*, indicating the value of the *`text`* UNIX environment variable.
ProductDir	Represents the directory where the product is installed.

New in this release

New in version 1.0.x

February 2004:

- Information about locale-dependent data has been moved to the section "Adapter environment" on page 1.
- The section "Verb processing" on page 7 has been updated with information about the difference between a blank verb ASI for a remote enterprise bean versus for a Java object.
- The section "Connector file structure" on page 11 has been updated with additional names of files that are copied to your system during installation. If you have already installed the product, you do not need to re-install; the files were already copied to your system when you previously ran the Installer.
- The section "Configuring the startup file" on page 21 has been updated with a new sample file name.
- Information about methods and business object attributes in the section "Connector business object structure" on page 26 has been updated.
- Information about business object attributes and their properties, including application specific information (the `AppSpecificInfo` property), in the section "Attribute-level ASI" on page 29 has been updated.
- Information about running the `MusicCart` enterprise bean sample files is provided in the section "Running the adapter to invoke sample business objects" on page 33.
- Information about the `CLIENT JARPATH` variable of the startup file has been updated in the section "Starting the ODA" on page 37.

December 2003:

- Version 1.0.x is the first release of the Adapter for EJB.

© Copyright IBM Corp. 2003, 2004

Chapter 1. Overview

This chapter provides an overview of the connector for EJB and contains the following sections:

The connector for EJB is a run time component of the WebSphere Business Integration Adapter for EJB. The EJB Adapter includes a connector, message files, configuration tools, and an Object Discovery Agent (ODA). The connector allows the WebSphere integration broker to exchange business objects with enterprise beans that have been designed using the Enterprise JavaBeans (EJB) architecture and deployed on an application server. The connector does this by enabling business processes in the broker to pass data to one or more enterprise bean business methods and receive returned data. The adapter is unidirectional in that it provides request processing functionality only. It does not perform event notification.

A connector consists of two components: the connector framework and the application-specific component. The connector framework, whose code is common to all connectors, acts as an intermediary between the integration broker and the application-specific component. The application-specific component contains code tailored to a particular technology (in this case, EJB) or application. The connector framework provides the following services between the integration broker and the application-specific component:

- Receives and sends business objects
- Manages the exchange of startup and administrative messages

This document contains information about both the connector framework and the application-specific component. It refers to both of these components as the connector.

Adapter environment

Before installing, configuring, and using the adapter, you must understand its environment requirements.

© Copyright IBM Corp. 2003, 2004

Broker compatibility

The adapter framework that an adapter uses must be compatible with the version of the integration broker (or brokers) with which the adapter is communicating. The 1.0.x version of the adapter for EJB is supported on the following versions of the adapter framework and with the following integration brokers:

- Adapter framework:
  - WebSphere Business Integration Adapter Framework, version 2.10, 2.2.0, 2.3.0, 2.3.1, 2.4.0
- Integration brokers:
  - WebSphere InterChange Server, version 4.1.1, 4.2.0, 4.2.1, 4.2.2
  - WebSphere MQ Integrator, version 2.1
  - WebSphere MQ Integrator Broker, version 2.1.0
  - WebSphere Business Integration Message Broker, version 5.0
  - WebSphere Application Server Enterprise, version 5.0.2, with WebSphere Studio Application Developer Integration Edition, version 5.0.1

See *Release Notes* for any exceptions.

Note: For instructions on installing the integration broker and its prerequisites, see the following documentation.

For WebSphere InterChange Server (ICS), see the *System Installation Guide for UNIX* or *for Windows*.

For message brokers (WebSphere MQ Integrator Broker, WebSphere MQ Integrator, and WebSphere Business Integration Message Broker), see *Implementing Adapters with WebSphere Message Brokers*, and the installation documentation for the message broker. Some of this can be found at the following Web site:

http://www.ibm.com/software/integration/mqfamily/library/manualsa/

For WebSphere Application Server, see Implementing Adapters with WebSphere Application Server and the documentation at

http://www.ibm.com/software/webservers/appserv/library.html

Adapter standards

The adapter is written to the EJB 2.1 specification and as such is compatible with a J2EE-compliant application server that is based on this standard, such as WebSphere Application Server, version 5.0.

The adapter supports entity beans and session beans. Message-driven beans are not supported in this release.

The adapter is compliant with the Java Authentication and Authorization Service (JAAS), release 1.0.

For information about EJB software architecture, see http://java.sun.com/products/ejb/

Adapter platforms

The adapter runs on the following platforms:

- Windows 2000
- Solaris 7, 8
- HP-UX 11i
- AIX(R) 5.1, 5.2

Adapter dependencies

The ODA for the EJB adapter requires the `j2ee.jar` file, which is delivered with the adapter.

Locale-dependent data

The connector has been internationalized so that it can support delivery of double-byte character sets going into an EJB interface that also supports double-byte character sets, and deliver message text in the specified language. When the connector transfers data from a location that uses one character code to a location that uses a different code set, it 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 WebSphere business integration system are written in Java; therefore, when data is transferred between most integration components, there is no need for character conversion.

Terminology

The following terms are used in this guide:

- **ASI (Application-Specific Information)** Metadata tailored to a particular application or technology. ASI exists at both the attribute and business object level of a business object. See also *Verb ASI*.
- **BO (Business Object)** A set of attributes that represent a business entity (such as Employee) and an action on the data (such as a create or update operation). Components of the WebSphere business integration system use business objects to exchange information and trigger actions.
- **BO (Business Object) handler** A connector component that contains methods that interact with an application and that transforms request business objects into application operations.
- **Deployment descriptor** An XML file, packaged in the JAR (Java Archive) file, that provides structural application assembly information about which classes make up the beans in the JAR file, how the beans should be deployed, and how each bean should be managed at runtime. When an enterprise bean is deployed to an enterprise application server, the deployment descriptor is read and the properties are displayed for editing. The person deploying the bean can then modify and add settings as needed. When the deployer is satisfied with the deployment information, he or she uses it to generate the entire supporting infrastructure needed to deploy the enterprise bean to the server.
- **EJB Container** Manages the interactions between a bean and the application server by providing life cycle management, security, deployment, and runtime services. This programmatic entity resides on the application server and as such, handles caching at the server-side rather than the client-side. The container

isolates the enterprise bean from direct access by client applications. When a client application invokes a remote method on an enterprise bean, the container first intercepts the invocation to ensure that persistence, transactions, and security are applied properly to every operation a client performs on the bean. When a bean is not being used, the container places it in a pool to be reused by another client, or possibly wipe it from memory while its remote reference on the client remains intact, only to bring it back when needed. When the client invokes a method on the remote reference, the EJB container simply resurrects the bean to service the request, with the client application being unaware of the entire process. An enterprise bean cannot function outside of an EJB container.

- **EJB home object** An object that provides the enterprise bean life cycle operations—create, remove, find. The EJB home object implements the enterprise bean's home interface, and the class for the EJB home object is generated by the container's deployment tools. The client uses the Java Naming and Directory Interface (JNDI) to locate an EJB home object. Then, the client references an EJB home object to perform life cycle operations on an EJB object.
- **Enterprise bean** An Enterprise JavaBeans component, that consists of a *home interface*, *remote interface*, and a *bean class*. The home interface represents the life cycle methods of the component (create, destroy, find), while the remote interface represents the business methods of the bean. The bean class implements the business methods defined in the remote interface, and as such is the key element of the bean.

  Enterprise beans are packaged in a JAR file, deployed to an enterprise application server, and managed by an EJB container. An EJB JAR file contains one or more enterprise beans and a deployment descriptor. An enterprise bean cannot function outside of an EJB container.
- **Enterprise bean class** Implements an enterprise bean's business methods.
- **Enterprise JavaBeans (EJB)** EJB enables rapid and simplified development of object-oriented, distributed, transactional, secure and portable enterprise-level Java applications. The EJB specification mandates a programming model; that is, it mandates conventions or protocols and a set of classes and interfaces which make up the EJB API. The EJB programming model provides enterprise bean and server developers with a common platform for development.
- **Entity bean** A bean that models a business concept that can be expressed as a noun. `Customer`, `Item`, and `Vendor` are examples of names of entity beans, since they model real-world objects. These objects are usually persistent records in a database. See also *session bean*.
- **Foreign key** A simple attribute whose value uniquely identifies a child business object. Typically, this attribute identifies a child business object to its parent by containing the child's primary key value. The connector for EJB uses the foreign key to specify poolable connection objects.
- **Home interface** Defines the bean's life cycle methods: methods for creating, removing, and finding the enterprise bean's remote interface, as well as home business methods, which are the business methods that are not specific to a particular bean instance.
- **Java Authentication and Authorization Service (JAAS)** A security framework that enables services to authenticate and enforce access controls based on user identity. It implements a Java version of the standard Pluggable Authentication Module (PAM) framework, and supports user-based authorization. JAAS authorization allows you to grant permissions based not on just what code is running but also on who is running it.
- **Java Naming and Directory Interface (JNDI)** A standard Java extension that provides a uniform API for accessing a wider range of services. The connector

for EJB uses JNDI to locate enterprise beans deployed on an enterprise application server. The server is required to support JNDI by organizing enterprise beans into a directory structure and providing a JNDI driver, called a service provider, for accessing that directory structure.

- **ODA (Object Discovery Agent)** A tool that automatically generates a business object definition by examining specified entities within the application and "discovering" the elements of these entities that correspond to business object attributes. When you install the adapter, the ODA is automatically installed. Business Object Designer provides a graphical user interface to access the ODA and to work with it interactively.
- **Per-call object pool** A programmatic entity for storing objects that need to pass from one method to the next during a single `doVerbFor` method call. Stored objects may be objects (child or non-child) or simple attributes.
- **Remote interface** Defines the bean's business methods that a client may call: the methods a bean presents to the outside world to do its work. A remote interface focuses on the business problem and does not include methods for system-level operations, such as persistence, security, concurrency or transactions. During connector processing, the connector accesses the enterprise bean through its remote object definition or stub.
- **Session bean** An enterprise bean that is created by a client as an extension of the client application. As such, a session bean is responsible for managing tasks and processes for the client, such as calculations or access to a database, and usually exists only for the duration of a single client-server session. Although a session bean may be transactional, it is not recoverable should a system crash occur. Session bean objects can be either stateless or can maintain conversational state across methods and transactions. If a session bean maintains state, then the EJB container manages this state if the object must be removed from memory; however, the session bean object itself must manage its own persistent data. See also *entity bean*.
- **Verb ASI (application-specific information)** For a given verb, the verb ASI specifies how the connector should process the business object when that verb is active. The verb ASI contains the name of the method to call to process the current request business object.

Request processing in the EJB adapter

This section describes how the connector for EJB handles request processing, as illustrated in Figure 1 on page 6.

The request processing scenario described here assumes that:

- The enterprise beans are deployed on a J2EE-compliant enterprise application server, such as WebSphere Application Server.
- The application server is installed and running.
- The connector has been initialized, which loads the `ProviderURL` and `InitialContextFactory` connector-specific properties. These properties help to obtain the JNDI initial context, which is required to initiate the connection to the EJB server and locate the enterprise bean's home interface. The properties are cached and re-used throughout the connector life cycle. For details about connector-specific properties, see "Connector-specific properties" on page 14. For details about how the connector manages security, see "Configuring security" on page 18.

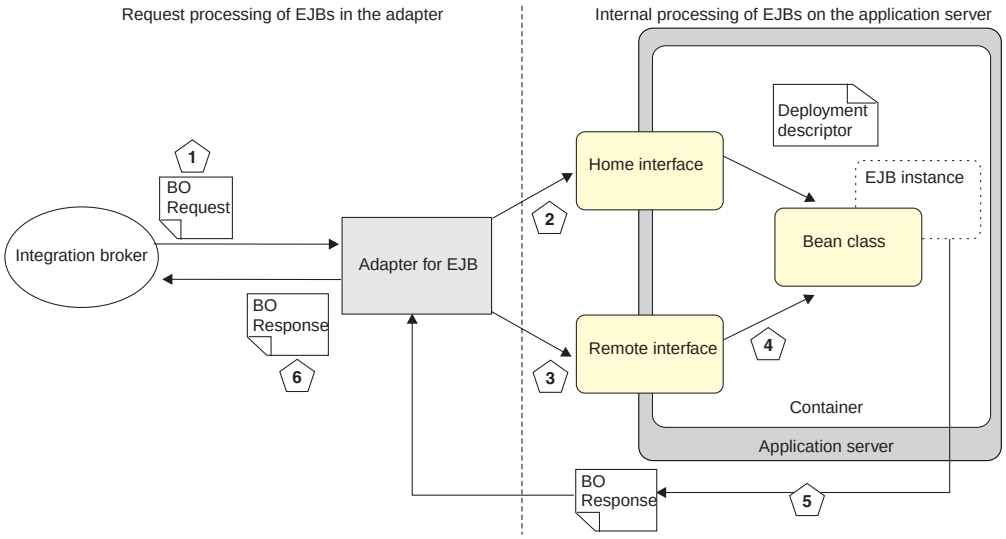


Figure 1. Request processing for the connector for EJB

The connector for EJB processes business object requests in the following manner.

1. The connector receives a business object request from the integration broker. The business object contains information about the corresponding enterprise bean, including its JNDI name and the class name of its home and remote interfaces.
2. The connector uses its `InitialContextFactory` and `ProviderURL` properties to begin the process of locating and accessing the enterprise beans deployed on the application server. `InitialContext` is part of the larger JNDI API supported by the application server. It is the starting point for any JNDI lookup by a client (such as the connector). The `ProviderURL` specifies the service provider, which is a JNDI driver on the application server used for locating the enterprise bean's home interface. The home interface provides methods for creating or finding the enterprise bean remote interface.
3. After locating the home interface, the connector finds the remote interface, which defines the enterprise bean business methods that a client (in this case, the connector) can call.

   For entity beans, the connector uses either a creator or finder type method, depending on which has been implemented by the enterprise bean developer (the connector knows which one to use based on the metadata in the business object definition). These methods are defined in the home interface and invoked by a client to obtain a remote/local interface reference to an enterprise bean instance.
4. Once the connector has located the remote interface, it can begin to invoke enterprise bean methods. The parent business object sent by the connector contains a child business object for each method defined in the remote interface. The child business object's attributes are mapped to parameters of the remote method of the corresponding enterprise beans. Since enterprise beans are loaded dynamically, its methods are discovered and called through reflection. The details of this step are as follows:
   - The connector's BO Handler checks the verb-level ASI of the parent business object for one ore more lists of attribute names. A list is a series of ordered, semi-colon-delimited attribute names.

- Each attribute of the parent business object contains a child business object that represents a method to be invoked on the remote interface. In other words, the verb ASI is not a list of methods, but a list of attributes, each one having as a value a child object that represents a method to be invoked.
- The connector executes the methods listed in the verb ASI, in the order in which they are listed.

5. After the methods are executed and values are returned from the EJB application server, the connector loads the EJB object data into the business object.
6. The connector returns the business object with populated values from the EJB application server to the integration broker.

   The connector also returns a message to the integration broker indicating that the original object request was either successful or unsuccessful (a FAIL status). If the request was successful, the connector also returns the updated business object to the broker.

EJB transaction support

Note that the connector does not support transacted invocations of EJB methods. If a business process requires transactions, create a thin wrapper bean that exposes one non-transactional method to the connector, but internally invokes the target EJB methods as part of a transaction in the application server.

Security for the adapter for EJB

Although it is an optional feature, when EJB security is configured in the connector, it requires that the connector provide authentication and access control data to the application server, before it can access any secure beans deployed on the server. The connector for EJB uses the Java Authentication and Authorization Service (JAAS) to implement EJB security.

JAAS is a security framework that enables services to authenticate and enforce access controls based on user identity. It implements a Java version of the standard Pluggable Authentication Module (PAM) framework, and supports user-based authorization. JAAS authorization allows you to grant permissions based not on just what code is running but also on who is running it.

JAAS authentication is performed in a pluggable fashion. This allows Java applications to remain independent from underlying authentication technologies, allowing new or updated authentication technologies to be plugged under an application without requiring modifications to the application itself.

For details about security, see “Configuring security” on page 18.

Verb processing

The connector processes business objects passed to it by a broker based on the verb for each business object.

When the connector framework receives a request from the broker, it calls the `doVerbFor()` method of the business-object-handler class associated with the business object definition of the request business object. The role of the `doVerbFor()` method is to determine the verb processing to perform, based on the active verb of the request business object. It obtains information from the request business object to build and send requests for operations to the application.

When the connector framework passes the request business object to `doVerbFor()`, this method retrieves the business object ASI and invokes the BO handler, which in turn reads the verb ASI and translates it into a series of callable functions. The verb ASI is an ordered list of the methods that need to be called for that verb. The order in which the calls are made is critical to the successful processing of the object.

If the verb ASI is blank for a remote enterprise bean, then the BO handler searches for one creator method (`Create()`) from the home interface with zero arguments, calls that, and then invokes the first method with populated parameters. The attribute-level ASI for the constructor should be `method_name=create`.

If the verb ASI is blank for a Java object, then the BO handler searches for one constructor and one method with populated parameters. The attribute-level ASI for the constructor should be `method_name=CONSTRUCTOR`.

On both a remote enterprise bean and a Java object, only one method can be populated; otherwise, if multiple methods are populated yet the verb ASI is blank, the connector logs an error and returns a FAIL code. For details about error processing, see "Error handling" on page 47. For details about verb ASI, see "Verb ASI" on page 28.

The connector does not support any specific verbs, but using the ODA, the user can configure custom verbs. The standard, pre-existing verbs are Create, Retrieve, Update, and Delete. These can be given whatever semantic meaning you provide through the Object Discovery Agent (ODA) running in Business Object Designer. For details about using the ODA to assign a method call sequence to a verb, see Chapter 5, "Creating and modifying business objects," on page 37.

Data handler processing

Using a data handler is optional. If you have designed a data handler into your connector architecture, the connector requires access to any data handler classes needed to convert business object values to enterprise bean parameters (as specified in the EJB ASI). An enterprise bean method can take an XML, EDI or other WBI data handler-supported document as an argument to a remote EJB method.

When the connector receives a business object, it will evaluate the business object ASI to determine if the business object needs to be converted into data using the data handler. The business object ASI for a data handler-supported message should contain the value `object_type=dataHandlerObject; mime_type=<text_value>` where `<text_value>` is the appropriate mime-type defined for the data handler (as specified in the data handler meta-object) that the adapter should use to convert the data.

If the connector finds a method business object with a data handler-supported document as a parameter, then the connector calls the data handler to convert the business object to the corresponding document, and then invokes a remote enterprise bean method by passing the document generated by the data handler as an argument to the method. Similarly if a method returns a document that must be processed using the data handler, the connector converts the String returned by the method into a business object using data handler values. For example, if an enterprise bean method returns an XML or EDI document, the data handler must be invoked to convert it into a child business object.

To support a data handler, you must configure the `DataHandlerConfigMO` connector-specific property. For details about this and other connector-specific properties, see "Connector-specific properties" on page 14. For details about developing a data handler, see the *Data Handler Guide*.

Chapter 2. Installing the adapter

This chapter describes how to install the connector and contains the following sections:

- "Overview of installation tasks"
- "Connector file structure"
- "Post-installation tasks" on page 12

Overview of installation tasks

To install the adapter for EJB, you must confirm that the necessary adapter prerequisites exist in your environment, install the integration broker, and run the adapter installation.

Confirm adapter prerequisites

Before you install the adapter, confirm that all the environment prerequisites for installing and running the adapter are on your system. For details, see "Adapter environment" on page 1.

Install the integration broker

Installing the integration broker, a task that includes installing the WebSphere business integration system and starting the broker, is described in the documentation for your broker. For details about the brokers that the connector for EJB supports, see "Broker compatibility" on page 2.

For details about installing the broker, see the appropriate implementation documentation of the broker you are using.

Install the adapter for EJB and related files

For information on installing WebSphere Business Integration adapter products, refer to the *Installation Guide for WebSphere Business Integration Adapters*, located in the WebSphere Business Integration Adapters Infocenter at the following site:

http://www.ibm.com/websphere/integration/wbiadapters/infocenter

Connector file structure

The Installer copies the standard files associated with the connector into your system. It installs the connector into the *ProductDir*\connectors\EJB directory, and adds a shortcut for the connector to the Start menu. Note that *ProductDir* represents the directory where the connector is installed.

Table 1 describes the file structure used by the connector, and shows the files that are automatically installed when you choose to install the connector through the Installer.

Table 1. File structure for the connector

Subdirectory of *ProductDir*	Description
\connectors\EJB\BIA_EJB.jar	Contains classes used by the EJB connector only
\connectors\EJB\start_EJB.bat	The startup script for the generic connector (Windows 2000)

© Copyright IBM Corp. 2003, 2004

Table 1. File structure for the connector (continued)

Subdirectory of *ProductDir*	Description
\connectors\EJB\start_EJB.sh	The startup script for the generic connector (Unix)
\connectors\messages\BIA_EJBConnector.txt	Message file for the connector
\connectors\EJB\samples\BIA_EJBConnector.cfg	Sample EJB connector configuration file. For details about using this file, see "Running the adapter to invoke sample business objects" on page 33.
\connectors\EJB\samples\BIA_PortConnector.cfg	Sample Port connector configuration file. For details about using this file, see "Running the adapter to invoke sample business objects" on page 33.
\connectors\EJB\samples\SampleBOs\	Sample business object definitions. For details about using these files, see "Running the adapter to invoke sample business objects" on page 33.
\connectors\EJB\samples\SampleMusicCartEJB\src	The EJB source files for the sample beans. For details about using the samples, see "Running the adapter to invoke sample business objects" on page 33.
\connectors\EJB\samples\SampleMusicCartEJB\ BIA_MusicBeanSample.jar	The EJB JAR file that contains the enterprise beans you deploy to the application server when running the sample files. For details about using the samples, see "Running the adapter to invoke sample business objects" on page 33.
\repository\EJB\EJBConnectorTemplate	Template file for the connector definition.
\connectors\EJB\dependencies\j2ee.jar	A JAR file required by the EJB ODA.
\ODA\EJB\BIA_EJBODA.jar	The EJB ODA
\ODA\EJB\start_EJBODA.bat	The ODA startup file (Windows 2000)
\ODA\EJB\start_EJBODA.sh	The ODA startup file (Unix)
\ODA\messages\BIA_EJBODAAgent.txt	Message file for the ODA
\ODA\messages\BIA_EJBODAAgent_de_DE.txt	Message file for the ODA (German text strings)
\ODA\messages\BIA_EJBODAAgent_en_US.txt	Message file for the ODA (US English text strings)
\ODA\messages\BIA_EJBODAAgent_es_ES.txt	Message file for the ODA (Spanish text strings)
\ODA\messages\BIA_EJBODAAgent_fr_FR.txt	Message file for the ODA (French text strings)
\ODA\messages\BIA_EJBODAAgent_it_IT.txt	Message file for the ODA (Italian text strings)
\ODA\messages\BIA_EJBODAAgent_ja_JP.txt	Message file for the ODA (Japanese text strings)
\ODA\messages\BIA_EJBODAAgent_ko_KR.txt	Message file for the ODA (Korean text strings)
\ODA\messages\BIA_EJBODAAgent_pt_BR.txt	Message file for the ODA (Portuguese Brazil -text strings)
\ODA\messages\BIA_EJBODAAgent_zh_CN.txt	Message file for the ODA (Simplified Chinese text strings)
\ODA\messages\BIA_EJBODAAgent_zh_TW.txt	Message file for the ODA (Traditional Chinese text strings)
\repository\EJB\EJBConnectorTemplate	Repository definition for the connector.

Note: All product pathnames are relative to the directory where the product is installed on your system.

Post-installation tasks

After installing the adapter, you must configure it before you can run it. For details, see Chapter 3, "Configuring the adapter," on page 13.

Chapter 3. Configuring the adapter

This chapter describes how to configure the adapter and contains the following sections:

Overview of configuration tasks

After installation and before startup, you must configure the connector and configure your business objects.

- **Configure the connector**: Configuring the connector includes setting up and configuring the connector. For details, see "Configuring the connector."
- **Configure the business objects**: You configure business objects through an ODA (Object Discovery Agent). The ODA enables you to generate business object definitions. A business object definition is a template for a business object. The ODA examines specified application objects, "discovers" the elements of those objects that correspond to business object attributes, and generates business object definitions to represent the information. Business Object Designer provides a graphical interface to access the Object Discovery Agent and to work with it interactively

  For details about using the ODA, see Chapter 5, "Creating and modifying business objects," on page 37.

Configuring the connector

Connectors have two types of configuration properties: standard configuration properties and connector-specific configuration properties. You must set the values of these properties using Connector Configurator before running the connector. For further information, see Appendix B, "Connector Configurator," on page 67.

A connector obtains its configuration values at startup. During a run time session, you may want to change the values of one or more connector properties. Changes to some connector configuration properties, such as `AgentTraceLevel`, take effect immediately. Changes to other connector properties require connector component restart or system restart. To determine whether a property is dynamic (taking effect immediately) or static (requiring either connector component restart or system restart), refer to the Update Method column in the Connector Properties window of the System Manager.

© Copyright IBM Corp. 2003, 2004

Standard connector properties

Standard connector configuration properties provide information that all connectors use. See Appendix A, "Standard configuration properties for connectors," on page 51 for documentation of these properties.

Note that although the following properties are listed in Appendix A, "Standard configuration properties for connectors," on page 51, the connector for EJB does *not* use these properties:

- `DuplicateEvent Elimination`
- `PollEndTime`
- `PollFrequency`
- `PollStartTime`

Also, note issues with the following properties:

- `Locale`: Because this connector has been internationalized, you can change the value of the `Locale` property
- `ApplicationName`: You must provide a value for the `ApplicationName` configuration property before running the connector.

Connector-specific properties

Connector-specific configuration properties provide information needed by the connector at run time. These properties also provide a way for you to change static information or logic within the connector without having to recode and rebuild it.

To configure connector-specific properties, use Connector Configurator. Click the **Application Config Properties** tab to add or modify configuration properties. For more information, see Appendix B, "Connector Configurator," on page 67.

Table 2 lists the connector-specific configuration properties, along with their descriptions and possible values. Note that these properties are all non-hierarchical Strings. See the sections that follow for details about the properties, including an image of the properties in Figure 2 on page 16.

Table 2. Connector-specific configuration properties

Name	Possible values	Default value
InitialContextFactory	The class name of the initial context factory.	`com.ibmwebsphere.naming.WsnInitialContextFactory`
ProviderURL	The URL of the JNDI service provider. The service provider is a driver, on the application server, that provides access to the directories in which the enterprise beans are stored on the server.	`corbaloc:iiop:localhost:2809`
DataHandlerConfigMO	The data handler meta-object. Used to support a data handler, if you have defined one for the connector.	`MO_DataHandler_Default`

Table 2. Connector-specific configuration properties (continued)

Name	Possible values	Default value
LoginConfiguration	The name of the LoginModule class for JAAS security.	The default value is determined by your application server. For WebSphere Application Server, the class is `WSLogin`. This property can only be used if your application server provides JAAS support, and you choose to implement secure access beans.
CallBackHandlerClass	The user-implemented CallBackHandler interface used for JAAS security.	For WebSphere Application Server, the class is `com.ibm.websphere.security.auth.callback.WSCallbackHandlerImpl` This property can only be used if your application server provides JAAS support and you choose to implement secure access beans.
JAASUserName	The JAAS security user name.	None Configure this property only if your application server provides JAAS support and you choose to implement secure access beans.
JAASPassword	The JAAS security password.	None Configure this property only if your application server provides JAAS support and you choose to implement secure access beans.
JAASRealm	The JAAS security realm name.	None Configure this property only if your application server provides JAAS support and you choose to implement secure access beans.

Figure 2 on page 16 illustrates the hierarchical relationship of the connector-specific properties.

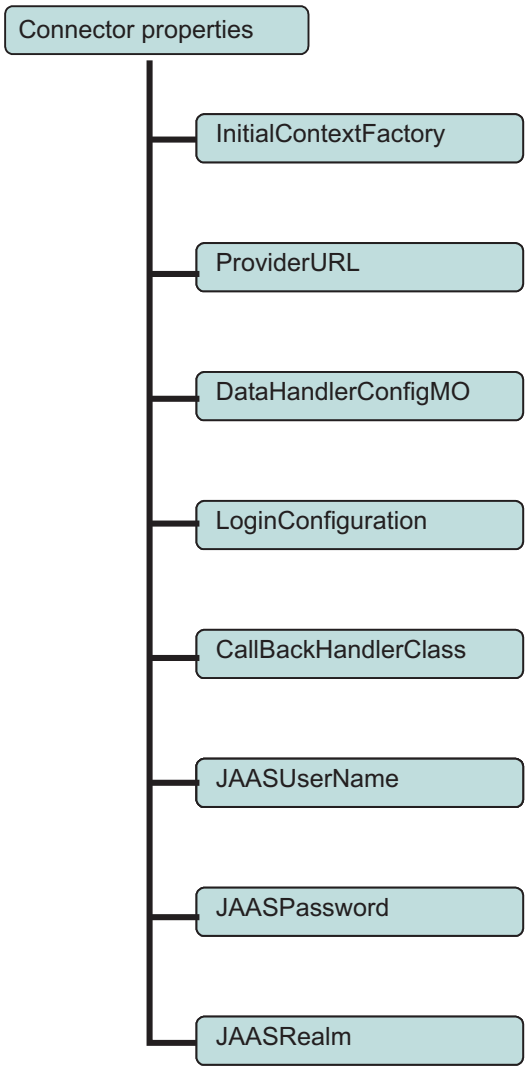


Figure 2. Hierarchy of connector-specific properties

InitialContextFactory

The class name of the initial context factory. `InitialContext`, a JNDI interface required for initiating the connection to the enterprise application server and locating the enterprise bean's home interface, is the starting point of any client lookup of an enterprise bean. The connector uses the `InitialContextFactory` property, along with the `ProviderURL` property, to obtain the JNDI initial context.

ProviderURL

The URL of the JNDI provider. JNDI enables the connector to access the enterprise beans by name.The connector uses this URL to connect remotely to the JNDI server running within the EJB server. After connecting to the EJB server, the connector can locate the home interface of the enterprise bean.

DataHandlerConfigMO

The name of the top-level data handler meta-object. If you have designed a data handler into your connector architecture, the connector requires access to any data handler classes needed to convert business object values to enterprise bean parameters (as specified in the EJB ASI). An enterprise bean method can take an

XML, EDI or other WBI data handler-supported document as an argument to a remote EJB method. If the connector finds a method business object with a data handler-supported document as a parameter, then the connector calls the data handler to convert the business object to the corresponding document, and then invokes a remote enterprise bean method by passing the document generated by the data handler as an argument to the method.

If this property is not filled in, the connector cannot find the meta-object, which it needs to invoke the appropriate data handler. For information about using a data handler with the connector, see "Data handler processing" on page 8.

LoginConfiguration

The class that implements the `LoginModule` interface of the authentication technology provider. The authentication technology provider implements the `LoginModule` to provide a particular kind of authentication via a pleadable module, without requiring any modifications to the application itself. Typically, the `LoginConfiguration` and `LoginModule` implementation are provided in the application server's login class. If your application server supports secure access beans using JAAS, and you choose to implement this service for EJB security, set this property to the name of the LoginModule class of your application server.

For JAAS-compliant application servers, the connector enables the authentication process by instantiating a LoginContext object, which specifies the LoginModules that will handle JAAS authentication. During authentication of the client, the LoginModule prompts for and verifies the username and password, defined in the `JAASUserName` and `JAASPassword` properties.

Although JAAS is not a requirement of the connector, if you specify a value in the `LoginConfiguration` property, the connector assumes you are implementing bean security, in which case the `CallBackHandlerClass`, `JAASUserName`, `JAASPassword`, and `JAASRealm` properties are all required.

For details about configuring security, see "Configuring security" on page 18 and "Security for the adapter for EJB" on page 7.

CallBackHandlerClass

A JAAS interface, determined by the application server, that enables a client to pass authentication data to the application server. This interface implements a CallBack handler that is passed to the underlying security services so that they can interact with the application to retrieve from the client (in this case, the connector) certain authentication data, such as username and password. The `LoginModule` uses the CallBack handler to communicate with the client to obtain the requested authentication data. Username and password for the connector are defined in the `JAASUserName` and `JAASPassword` properties.

If your application server supports secure access beans using JAAS, and you choose to implement this service for EJB security, set this value to the user-implemented `CallBackHandler` interface.

Although JAAS is not a requirement of the connector, if you specify a value in the `LoginConfiguration` property, the connector assumes you are implementing bean security, in which case the `CallBackHandlerClass`, `JAASUserName`, `JAASPassword`, and `JAASRealm` properties are all required.

For details about configuring security, see "Configuring security" on page 18 and "Security for the adapter for EJB" on page 7.

JAASUserName

If your application server supports secure access beans using JAAS, and you choose to implement this service for EJB security, set this value to the JAAS security user name configured on your application server.

Although, JAAS is not a requirement of the connector, to properly configure security using JAAS, you must specify values in the `JAASUserName`, `LoginConfiguration`, `CallBackHandlerClass`, `JAASPassword`, and `JAASRealm` properties.

For details about configuring security, see "Configuring security" and "Security for the adapter for EJB" on page 7.

JAASPassword

If your application server supports secure access beans using JAAS, and you choose to implement this service for EJB security, set this value to the JAAS security password configured on your application server.

Although, JAAS is not a requirement of the connector, to properly configure security using JAAS, you must specify values in the `JAASPassword`, `LoginConfiguration`, `CallBackHandlerClass`, `JAASUserName`, and `JAASRealm` properties.

For details about configuring security, see "Configuring security" and "Security for the adapter for EJB" on page 7.

JAASRealm

If your application server supports secure access beans using JAAS, and you choose to implement this service for EJB security, set this value to the JAAS security realm name. A realm is a JAAS mapping of one or more User Groups to a set of privileges or permissions.

Although, JAAS is not a requirement of the connector, to properly configure security using JAAS, you must specify values in the `JAASRealm`, `LoginConfiguration`, `CallBackHandlerClass`, `JAASUserName`, and `JAASPassword` properties.

For details about configuring security, see "Configuring security" and "Security for the adapter for EJB" on page 7.

Configuring security

EJB security in the connector is optional, and only available if the application server on which the enterprise beans are deployed supports secure beans and the security protocol you choose. If you do enable security, it requires that the user (in this case, the connector) provide authentication data to the application server to access any deployed beans. Security can be enabled in the EJB connector using the Java Authentication and Authorization Service (JAAS) only if your application server supports this security technology.

Assigning method permissions to security roles

The connector relies on access control to ensure that users only access resources for which they have been given permission. Access control applies security policies that regulate what a specific user can and cannot do within a system. The security roles for the enterprise beans in the EJB JAR file specify the methods of the home

and remote interfaces that each security role is allowed to invoke. The deployment descriptor file includes tags that declare which logical roles are allowed to access which bean methods at runtime.

The following sample code illustrates how security roles are assigned method permissions in the deployment descriptor. Notice that for each role listed in the deployment descriptor, the methods that can be invoked are grouped by bean name.

```
<method-permission>
        <role-name>payroll_department</role-name>
        <method>
                <ejb-name>EmployeeService</ejb-name>
                <method-name>*</method-name>
        </method>
</method-permission>
<method-permission>
        <role-name>employee</role-name>
        <method>
                <ejb-name>AardvarkPayroll</ejb-name>
                <method-name>findByPrimaryKey</method-name>
        </method>
        <method>
                <ejb-name>AardvarkPayroll</ejb-name>
                <method-name>getEmployeeInfo</method-name>
        </method>
        <method>
                <ejb-name>AardvarkPayroll</ejb-name>
                <method-name>updateEmployeeInfo</method-name>
        </method>
</method-permission>
<method-permission>
```

Using the Java Authentication and Authorization Service (JAAS)

If your application server supports the Java Authentication and Authorization Service (JAAS), you can use it to implement EJB security using this service. When the user or client (in this case, the connector) attempts to access a secure bean deployed on an application server, JAAS verifies the user and that user's permissions. In doing so, JAAS provides an additional level of security to the application server on which an enterprise bean is deployed. JAAS requires specifying a `LoginContext`, `LoginModule`, and other J2EE classes that describe the basic methods used to authenticate a client. For additional information about these classes, see "LoginConfiguration" on page 17.

Since the connector for EJB acts as a client of the server, when security is enabled in the EJB environment, the connector must provide user authentication data in the form of a user ID (`JAASUserName` configuration property of the connector) and password (`JAASPassword` configuration property). The EJB container examines the identity or role passed by the connector, and compares it with the list of identity objects associated with the method as defined in the deployment descriptor. If the caller identity from the connector matches a caller identity associated with the method, then the method can be invoked.

Authentication validates the identity of the user (in this case, the connector). Once the user has successfully passed through the authentication system, it is free to access and invoke the methods for which it has permissions.

The connector properties that control EJB security using JAAS are `LoginConfiguration`, `CallBackHandlerClass`, `JAASUserName`, `JAASPassword`, and

JAASRealm. The values you enter in these properties are entirely dependent on your application server. For details about how the connector uses these properties, see "Connector-specific properties" on page 14. For details about the values to enter in the properties, see your application server documentation.

Configuring JAAS in the connector startup file

For the connector to work with JAAS, you must make the following changes to the connector startup file (start_EJB.bat or start_EJB.sh):

- Enter the following command in the startup file to specify the JAAS configuration file name and location. This command should appear as one of the last lines in the file.

  ```
  -Djava.security.auth.login.config=login.conf
  ```

  In the sample command provided here, the JAAS configuration file has the name login.conf; however, this name varies according to the application server environment. For details about the JAAS configuration file name and location for your environment, see your application server documentation.
- Uncomment the SECURITY_SETTINGS section in the connector startup file to use the JAAS API with the connector.

  Note that the JAAS API is available in the jaas.jar file provided by any EJB-supported application server. Make sure to reference jaas.jar in the classpath of the connector.

Creating multiple connector instances

Creating multiple instances of a connector is in many ways the same as creating a custom connector. You can set your system up to create and run multiple instances of a connector by following the steps below. You must:

- Create a new directory for the connector instance
- Make sure you have the requisite business object definitions
- Create a new connector definition file
- Create a new start-up script

Create a new directory

You must create a connector directory for each connector instance. This connector directory should be named:

```
ProductDir\connectors\connectorInstance
```

where connectorInstance uniquely identifies the connector instance.

If the connector has any connector-specific meta-objects, you must create a meta-object for the connector instance. If you save the meta-object as a file, create this directory and store the file here:

```
ProductDir\repository\connectorInstance
```

Create business object definitions

If the business object definitions for each connector instance do not already exist within the project, you must create them.

1. If you need to modify business object definitions that are associated with the initial connector, copy the appropriate files and use Business Object Designer to import them. You can copy any of the files for the initial connector. Just rename them if you make changes to them.
2. Files for the initial connector should reside in the following directory:

ProductDir\repository\initialConnectorInstance

Any additional files you create should be in the appropriate connectorInstance subdirectory of ProductDir\repository.

Create a connector definition

You create a configuration file (connector definition) for the connector instance in Connector Configurator. To do so:

1. Copy the initial connector's configuration file (connector definition) and rename it.
2. Make sure each connector instance correctly lists its supported business objects (and any associated meta-objects).
3. Customize any connector properties as appropriate.

Create a start-up script

To create a startup script:

1. Copy the initial connector's startup script and name it to include the name of the connector directory:

   dirname
2. Put this startup script in the connector directory you created in "Create a new directory" on page 20.
3. Create a startup script shortcut (Windows only).
4. Copy the initial connector's shortcut text and change the name of the initial connector (in the command line) to match the name of the new connector instance.

You can now run both instances of the connector on your integration server at the same time.

For more information on creating custom connectors, refer to the *Connector Development Guide for C++ or for Java*.

Configuring the startup file

Before you start the connector for EJB, you must configure the connector startup file.

To complete the configuration of the connector for Windows platforms, you must modify the start_EJB.bat file:

1. Open the start_EJB.bat file.
2. Scroll to the section beginning with "SET JCLASSES..."
3. Edit the JCLASSES variable to point to the JAR file created by the ODA. For example, if the JAR file created by the ODA is c:\WebSphereAdapters\connectors\EJB\SampleBeans.jar, then set the JCLASSES variable to: JCLASSES=.;%J_CLASSES%;c:\WebSphereAdapters\connectors\EJB\SampleBeans.jar

Starting the connector

A connector must be explicitly started using its **connector start-up script**. The startup script should reside in the connector's runtime directory:

ProductDir\connectors*connName*

where *connName* identifies the connector. The name of the startup script depends on the operating-system platform, as Table 3 shows.

Table 3. Startup scripts for a connector

Operating system	Startup script
UNIX-based systems	`connector_manager_`*`connName`*
Windows	`start_`*`connName`*`.bat`

You can invoke the connector startup script in any of the following ways:

- On Windows systems, from the **Start** menu

  Select **Programs>IBM WebSphere Business Integration Adapters>Adapters>Connectors**. By default, the program name is "IBM WebSphere Business Integration Adapters". However, it can be customized. Alternatively, you can create a desktop shortcut to your connector.
- From the command line
  - On Windows systems:

    `start_`*`connName connName brokerName`* `[-c`*`configFile`* `]`
  - On UNIX-based systems:

    `connector_manager_`*`connName`* `-start`

  where *connName* is the name of the connector and *brokerName* identifies your integration broker, as follows:
  - For WebSphere InterChange Server, specify for *brokerName* the name of the ICS instance.
  - For WebSphere message brokers (WebSphere MQ Integrator, WebSphere MQ Integrator Broker, or WebSphere Business Integration Message Broker) or WebSphere Application Server, specify for *brokerName* a string that identifies the broker.

  Note: For a WebSphere message broker or WebSphere Application Server on a Windows system, you *must* include the `-c` option followed by the name of the connector configuration file. For ICS, the `-c` is optional.
- From Adapter Monitor (WebSphere Business Integration Adapters product only), which is launched when you start System Manager

  You can load, activate, deactivate, pause, shutdown or delete a connector using this tool.
- From System Monitor (WebSphere InterChange Server product only)

  You can load, activate, deactivate, pause, shutdown or delete a connector using this tool.

For more information on how to start a connector, including the command-line startup options, refer to one of the following documents:

- For WebSphere InterChange Server, refer to the *System Administration Guide*.
- For WebSphere message brokers, refer to *Implementing Adapters with WebSphere Message Brokers*.
- For WebSphere Application Server, refer to *Implementing Adapters with WebSphere Application Server*.

Stopping the connector

The way to stop a connector depends on the way that the connector was started, as follows:

- If you started the connector from the command line, with its connector startup script:
  - On Windows systems, invoking the startup script creates a separate "console" window for the connector. In this window, type "Q" and press Enter to stop the connector.
  - On UNIX-based systems, connectors run in the background so they have no separate window. Instead, run the following command to stop the connector:

    `connector_manager_connName -stop`

    where *connName* is the name of the connector.
- From Adapter Monitor (WebSphere Business Integration Adapters product only), which is launched when you start System Manager

  You can load, activate, deactivate, pause, shutdown or delete a connector using this tool.
- From System Monitor (WebSphere InterChange Server product only)

  You can load, activate, deactivate, pause, shutdown or delete a connector using this tool.

Using log and trace files

The connector components provide several levels of message logging and tracing. The connector uses the adapter framework to log error, informational, and trace messages. Error and informational messages are recorded in the log file, and trace messages and their corresponding trace levels (0 to 5) are recorded in a trace file. For details about logging and trace levels, see Chapter 6, "Troubleshooting and error handling," on page 47.

You configure both the log and trace file names, as well as the trace level, in Connector Configurator. For details about this tool, see Appendix B, "Connector Configurator," on page 67.

Note that the ODA has no logging capability. Error messages are sent directly to the user interface. Trace files and the trace level are configured in Business Object Designer. The process is described in "Configure the agent" on page 38. The ODA trace levels are the same as the connector trace levels, defined in "Tracing" on page 48.

Chapter 4. Understanding business objects

This chapter describes the structure of business objects, how the connector processes the business objects, and the assumptions the connector makes about them.

The chapter contains the following sections:

Defining metadata

The connector for EJB is metadata-driven. In the WebSphere business integration system, metadata is defined as application-specific information that describes the data structures of an object. The metadata is used to construct business object definitions, which the connector uses at run time to build business objects.

After installing the connector, but before you can run it, you must create the business objects definitions. The business objects that the connector processes can have any name allowed by the integration broker. For information about naming conventions, see *Naming Components Guide*.

A metadata-driven connector handles each business object that it supports according to the metadata encoded in the business object definition. This enables the connector to handle new or modified business object definitions without requiring modifications to the code. New objects can be created through the Object Discovery Agent (ODA) in Business Object Designer. To modify an existing object, use Business Object Designer directly (without going through the ODA).

Application-specific metadata includes the structure of the business object and the settings of its attribute properties. Actual data values for each business object are conveyed in message objects at run time.

The connector makes assumptions about the structure of its supported business objects, the relationships between parent and child business objects, and the format of the data. It is important that the structure of the business object exactly match the structure defined for the corresponding enterprise bean or the connector will not be able to process business objects correctly.

If you need to make changes to the business object structure, make them to the corresponding enterprise bean and then export the changes to the JAR file for input into the ODA.

For more information on modifying business object definitions, see *WebSphere Business Integration Adapters Business Object Development Guide.*

© Copyright IBM Corp. 2003, 2004

Connector business object structure

The connector processes business objects used by enterprise beans. This section describes the key concepts related to the structure of business objects processed by the EJB connector.

Methods

For each method defined in a Java class file, the ODA creates an attribute in the business object (for more information, see "Attributes" and "Attribute-level ASI" on page 29).

The type of the attribute that the ODA creates from a method is a child business object containing further attributes that represent method parameters and return types. These attributes appear in the exact same order as the parameters of the EJB method. The `Return_Value` attribute (not used if the method being defined is of type `void`) always appears last in the order of arguments, and represents the result of the EJB method call. Child business object attributes can be simple type or object type (complex), depending on the type of the method parameter or return value. The application specific information (ASI) for such attributes contains the exposed remote method name. For details about attribute ASI, see "Attribute-level ASI" on page 29.

The EJB connector requires a business object to have a creator method, obtained from the list of home interface creator methods, and one or more business object methods that have been specified in the remote interface. An attribute is created in the business object for each `creator` method on the home interface and for each method defined in the remote interface.

Whenever properties or method names contain special characters, the attribute names corresponding to these are modified to suit WebSphere Business Integration format.

Attributes

For each public member variable, attribute, and method parameter in an enterprise bean method that has been defined in a Java class file, a corresponding business object attribute is generated by the ODA. The JAR file, which contains the interfaces, is used by the ODA to create business object definitions that map to remote and local enterprise beans.

If an attribute in the bean class is not a simple attribute, but instead is an object, then the business object (BO) attribute maps to a child object whose definition matches the corresponding Java class or EJB interface.

Business objects can be flat or hierarchical. A flat business object only contains simple attributes, that is, attributes that represent a single value (such as a string) and do not point to child business objects. A hierarchical business object contains both simple attributes and child business objects or arrays of child business objects that contain attribute values.

A cardinality 1 container object, or single-cardinality relationship, occurs when an attribute in a parent business object contains a single child business object. In this case, the child business object represents a collection that can contain only one record. The attribute type is the child business object.

A cardinality `n` container object, or multiple-cardinality relationship, occurs when an attribute in the parent business object contains an array of child business objects. In this case, the child business object represents a collection that can contain multiple records. The attribute type is the same as that of the array of child business objects.

For additional information about business object attributes, see "Attribute-level ASI" on page 29.

Application-specific information

Application-specific information provides the connector with application-dependent instructions on how to process business objects. If you extend or modify a business object definition, you must make sure that the application-specific information in the definition matches the syntax that the connector expects.

Application-specific information is represented as a name-value pair and can be specified for the overall business object, for each business object attribute, and for each verb.

Business object-level ASI

Object-level ASI provides fundamental information about the nature of a business object and the objects it contains. To view the ASI of a business object, open the business object in Business Object Designer and click the General tab. The business object-level ASI displays in the **Business Object Level Application-specific information** field. For details about this screen, see Chapter 5, "Creating and modifying business objects," on page 37.

Table 4 describes the business object-level ASI of business objects that represent enterprise beans.

Note: ASI names are not recognized for business objects that represent methods, method parameters, and method return values. For details about business object attributes created for methods of enterprise beans, see "Methods" on page 26.

Table 4. Business object-level ASI

Object-level ASI	Description
`object_type=RemoteEJB`	Indicates that the object is an enterprise bean deployed on an application server (remote), rather than a standard Java class that is not deployed on an application server (local).
`jndi_name=<`*`JNDIName`*`>`	The JNDI name of the enterprise bean
`home_name=<`*`className`*`>`	The class name of the home interface
`proxy_class=<`*`className`*`>`	The class name of the remote interface

The following example illustrates business object-level ASI for a business object that the connector is processing with enterprise beans deployed on WebSphere Application Server. When the connector receives the business object from the broker, it first locates an instance of the home interface (com.ibm.websphere.AccountBookHome) on the application server, by looking up `WsSamples/Account` in the configured JNDI context. The connector than uses this home instance to create a new instance of the remote interface (`com.ibm.websphere.AccountBook`), which it can use to invoke methods on the EJB.

```
BO ASI=object_type = RemoteEJBObject
           proxy_class = com.ibm.websphere.AccountBook
           home_class = com.ibm.websphere.AccountBookHome
           jndi_name = WsSamples/Account
```

In this code sample:

- `BO ASI = object type` is required to show that the object is a a remote object.
- `proxy_class = com.ibm.websphere.AccountBook` represents the name of the remote stub.
- `home_class = com.ibm.websphere.AccountBookHome` is the name of the home interface.
- `jndi_name = WsSamples/Account` is the JNDI name.

A standard Java object (local) can have a `BO ASI = auto_load_or_write`.

The business object ASI for a data handler-supported message should contain the value `object_type=dataHandlerObject; mime_type=<text_value>` where *<text_value>* is the appropriate mime-type defined for the data handler (as specified in the data handler meta-object) that the adapter should use to convert the data.

Verb ASI

Every business object contains a verb. A verb indicates which methods to invoke on the enterprise bean. For the adapter for EJB, the first method should be a creator method from the corresponding bean's home interface, and the remaining methods should be business process methods from the bean's remote interface.

The verb ASI contains a sequence of attribute names, each of which contains a method for the business object handler to call. Typically, the method to be invoked belongs to the object itself (versus belonging to a parent of the business object), in which case you specify the method in the object's verb ASI.

If the method to be invoked belongs to a parent in the business object hierarchy, then that parent can be referenced from the child by prefixing the method name with the `PARENT` tag.

For example, Figure 3 on page 29 illustrates a business object hierarchy whereby ContactDetails is a child object of Contact, which itself is a child of PSRCustomerAccount.

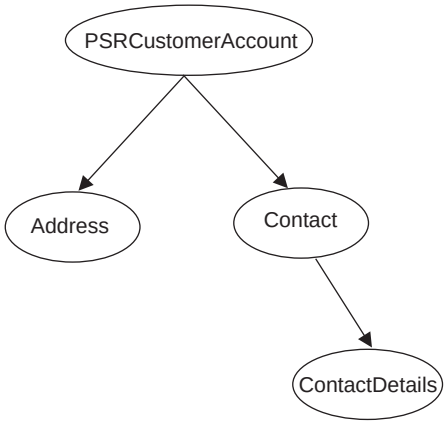


Figure 3. Business object hierarchy and verb ASI

If a method that belongs to PSRCustomerAccount is called on the ContactDetails business object, then the verb ASI for ContactDetails represents the business object hierarchy as follows:

`PARENT.PARENT.<methodName>`

If the method belongs instead to the Contact business object, then the verb ASI for ContactDetails must be set as:

`PARENT.<methodName>`

Note that only methods that belong to parent objects within the hierarchy can be called. A parent business object cannot invoke a child's method.

The connector developer determines the EJB operations assigned to the verb. Supported verbs include:

- Create
- Delete
- Retrieve
- Update

For a given object, you can specify the four supported verbs (Create, Retrieve, Delete, and Update) and assign as actions of each verb `n` methods, where `n` equals the number of methods in the corresponding enterprise bean.

To view the verb ASI of a business object, open a business object in Business Object Designer and click the General tab. The verb ASI appears in the **Supported verbs** table, which lists the name of the supported verbs and the corresponding ASI for each verb. For more information about Business Object Designer and the ODA wizard tabs and screens, see Chapter 5, "Creating and modifying business objects," on page 37.

Attribute-level ASI

Every business object has a set of attributes, as described in "Attributes" on page 26, that map to corresponding enterprise bean methods and properties. A business object with a flat structure contains simple attributes, that is, attributes that represent a single value (such as a string) and do not point to child business objects. A hierarchical business object contains both simple attributes and child business objects or arrays of child business objects that contain attribute values.

The ASI of a business object attribute can be for simple attributes and complex attributes, which contain child objects. For a complex attribute, the ASI varies, depending on whether the contained child is a property or a method of an object. The mapping of EJB constructs to the ODA-generated business objects is described in Table 8 on page 32.

To view the ASI of a business object attribute, open a business object in Business Object Designer and click the Attributes tab. Each attribute of the business object is listed in the **Name** column. The attribute properties described in Table 5 on page 30, Table 6 on page 30, and Table 7 on page 31, appear in the remaining columns. Included in these properties is the ASI, which appears in the **Application Specific Information** column. For details about this screen, see Chapter 5, "Creating and modifying business objects," on page 37.

Table 5 describes the business object properties of simple attributes. These properties include the ASI (AppSpecificInfo property) of the attribute. A simple attribute is always a non-child, for example a boolean, string, or integer value.

Table 5. BO attribute properties: for simple attributes

Attribute property	Description
Name	Specifies the business object attribute name.
Type	Specifies the business object attribute type. See Table 8 on page 32 for details about mapping EJB constructs to business object attribute types.
MaxLength	Not used.
IsKey	Each business object must have at least one key attribute, which you specify by setting the key property to true for an attribute. Note that this attribute is used by Business Object Designer, rather than by the connector.
IsForeignKey	Specifies that the connector should check whether or not the object must be stored in the per call object pool.
IsRequired	Not used.
AppSpecInfo	Holds the original Java type. This property is formatted as follows: property=<*propertyName*>, type=<*typeName*> property is the name of the EJB attribute or object member variable. Use this name-value pair to capture the original EJB object member variable name. type is the Java type of a simple attribute (the EJB object member variable). For example, type=java.lang.String See Table 8 on page 32 for details about mapping EJB constructs to business object attribute types.
DefaultValue	Not used.

Table 6 describes the elements for complex attributes containing child objects that are not methods. These elements include the ASI of the attribute.

Table 6. BO attribute properties: for non-method child object attributes

Attribute property	Description
Name	Specifies the business object attribute name.
type	The type of the contained object. Set to proxy if the type is a business object.
ContainedObjectVersion	Not used.
Relationship	Specifies that the child is a container attribute. Set to Containment.
IsKey	Not used
IsForeignKey	Not used
Is Required	Not used

Table 6. BO attribute properties: for non-method child object attributes (continued)

Attribute property	Description
AppSpecificInfo	Holds the original EJB application field name. This property is formatted as follows: property=*propertyName*, use_attribute_value=<*(optional)BOName.AttributeName*>, type=<typeName> property is the name of the EJB object member variable. Use this name-value pair to capture the original EJB object attribute name. In the case of an attribute that holds an argument to a method, do not set a value for property, as the argument does not have a name and is simply an argument of any standard type. use_attribute_value is the business object name formatted as *BOName.AttributeName.* Setting this ASI causes the connector to access the attribute from the per call object pool. Note that this value is not set in the ODA when you create the business object, but rather via Business Object Designer. type is the Java type of a property. See Table 8 on page 32 for the mapping of EJB constructs to business objects.)
Cardinality	Set to *n* if the type represents an array or vector. Otherwise, set to 1.

Table 7 describes the properties of complex attributes containing child objects that are methods. These properties include the ASI of the attribute.

Table 7. BO attribute properties: for method child object attributes

Attribute property	Description
Name	The business object attribute name.
type	The business object.
Relationship	Set to Containment, indicating that this is a child object.
IsKey	Set to true if the attribute name equals UniqueName, otherwise it is set to false.
IsForeignKey	Set to false.
Is Required	Set to false.
AppSpecificInfo	Holds the exposed remote method interface name, which is the name of the method call placed to the EJB server by the enterprise bean. This attribute is formatted as: method_name=<*remoteClassName.RemoteMethodName*> If the method is a constructor, method_name=CONSTRUCTOR
Cardinality	Set to 1.

Note that methods have arguments and return values. Arguments and return values can be complex (containing child objects) or simple.

Mapping attributes: Enterprise JavaBeans (EJB) and business object

This section provides a list of the EJB constructs defined in a JAR file and their corresponding business object attributes. For all business object attributes that are not child business objects, the data type is `String`. In a business object, the ASI holds the actual data type of the attribute and is used when invoking methods against the enterprise bean's remote interface.

For details about business object ASI, see "Application-specific information" on page 27.

Table 8. Object mapping: Enterprise bean to business object

EJB Construct	Business object	Attribute ASI type=
All classes whose reference is found in the JAR file	Object	`proxy_class=<remote interface name>`
boolean	Boolean	`type=boolean/Boolean`
char/Character	String	`type=char/Character`
Byte/Byte	String	`type=byte/Byte`
java.lang.String	String	`type=string`
Short/Short	Integer	`type=short/Short`
int/Integer	Integer	`type=int/Integer`
Long/Long	Integer	`type=long/Long`
float/Float	Float	`type=float/Float`
Double/double	Double	`type=double/Double`
java.math.BigDecimal	String	`type=BigDecimal`
class	Object	`proxy_class=<fully qualified class name>`
array	Object Child business object with multiple cardinality	`type=ArrayOf_<datatype>` For example, `type=ArrayOf_int`
method	Object Child BO	`method_name=<methodName>`
method (no argument and void return type)	String	`method_name=<methodName>`

Note: In cases where the attribute is not intended to be de-referenced, the ASI `type=PlaceholderOnly` should be used. This tells the connector to not populate this attribute. The attribute may still be used as part of a multi-call flow if it is either marked as a foreign key (`IsForeignKey` is set to `true`), or has the ASI `use_attribute_value` pointing to a compatible attribute.

Array types

Note the following about `array` types:

- To use an `array` type, specify an ASI of `type=ArrayOf_<value>`, where *value* is one of the attribute ASI values listed in Table 8. For example, `type=ArrayOf_int` specifies an array of `int` variables. These are mapped to a cardinality n business object that contains the element.
- An Object array (`Object[]`) in Java has a corresponding ASI type of `ArrayOf_proxy`. The processing of proxy objects is done against every element of

the array. If the proxy array is an argument to a function, verb processing will occur on every object in the array *before* executing the method. If the array is a return value, verb processing will occur on every object in the array *after* executing the method.

- A sized array may be used as input but not as output.
- A SafeArray is supported as both an input and a return value.

Running the adapter to invoke sample business objects

The following topics describe running the sample files delivered with the product and provide examples that illustrate how the connector invokes the services of a JAR file to create business objects.

- "Running the sample files"
- "EJB JAR file code" on page 34
- "Business object sample" on page 35

Running the sample files

The sample files are located in the *ProductDir*\connectors\EJB\samples directory, where *ProductDir* represents the directory where the connector is installed.

The sample files illustrate a simple session bean, MusicCart, that provides a home interface, remote interface, bean and two helper classes, RecordingHelper and CustomerHelper. The sample files also include two connector configuration files: BIA_EJBConnector.cfg, which contains the connector properties settings for the example to run properly; and BIA_PortConnector.cfg, which contains the connector properties for the test connector.

The MusicCart sample bean has methods that take a Java object as an input parameter and return an object as output. It also has methods that take an object array as an input, illustrating how the adapter handles array management. The sample also demonstrates how the adapter performs data handler processing.

Included in the sample files is a business object file (BIA_SampleMusicCartBO.bo) and EJB source files (in *ProductDir*\connectors\EJB\samples\SampleMusicCartEJB\src), which show how EJB remote and home classes are mapped to business objects using the ODA for EJB. By sending BIA_SampleMusicCartBO.bo from the test connector to the application server, you can see how the addRecording, getFirstRecordInfo, modifyMusicRequestUsingDataHandler, and getAllRecordInfo methods are executed and how the connector handles request processing.

The following steps assume that you are running the connector to exchange business objects with enterprise beans deployed on WebSphere Application Server 5.0.

1. Install the adapter for EJB as described in Chapter 2, "Installing the adapter," on page 11.
2. Deploy the EJB JAR file *ProductDir*\connectors\EJB\samples\SampleMusicCartEJB\BIA_MusicBeanSample.jar to your instance of WebSphere Application Server 5.0.
3. Load the two connector configuration files in the *ProductDir*\connectors\EJB\samples\ directory (BIA_EJBConnector.cfg and BIA_PortConnector.cfg) into the repository of your integration broker, for example InterChange Server.

4. Load the sample business objects from the *ProductDir*\connectors\EJB\samples\SampleBOs directory into the broker repository.
5. If you want to install a remote client on a different machine than the one where your application server is running, install WebSphere Application Thin Client.
6. Modify the connector startup file to point to the location of the JAR file (BIA_MusicBeanSample.jar) you deployed in Step 2 on page 33. For information about editing the startup file, see "Configuring the startup file" on page 21.
7. In the startup file, uncomment the commands for your application server settings. For this sample, you must uncomment the commands for WebSphere Application Server, since this sample assumes you are running the connector to exchange business objects with enterprise beans deployed on WebSphere Application Server 5.0.
8. Start the WebSphere Application Server instance.
9. Start running the connector, as described in "Starting the connector" on page 21.
10. Send the file BIA_SampleMusicCartBO.bo from the test connector to the application server.

As the connector runs, you can observe how the enterprise bean methods defined in the sample files are executed from the adapter to invoke services provided by the remote WebSphere Application Server.

EJB JAR file code

The following sample code is an excerpt from the EJB JAR file that defines the methods of the EJB class called MusicCartBean. Notice the method defined at the end of the code sample: getCustomer.

Note: Not all of the methods for this class are defined in the code sample provided here. The sample is a section of a larger file described in "Running the sample files" on page 33.

The business object that corresponds to this code is illustrated in Figure 4 on page 35.

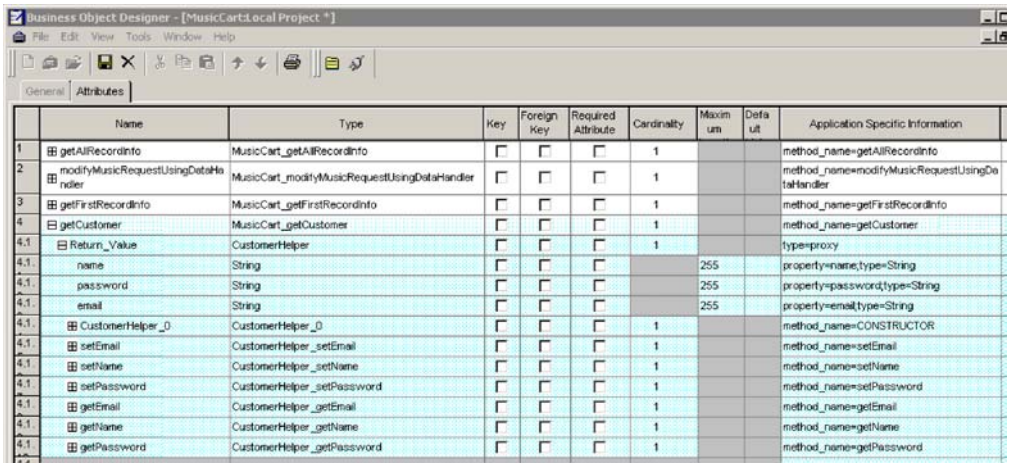
```
public class MusicCartBean implements SessionBean {
  CustomerHelper customerHelper;
  ArrayList shoppingList;
  RecordingHelper[] recordHelperArr;


   // EJB Methods
   public void ejbRemove() {}
   public void ejbActivate() {}
   public void ejbPassivate() {}
   public void setSessionContext (SessionContext ctx) {}

  public void ejbCreate(String person,String password, String email)
      throws CreateException {
           if (person == null || person.equals("")) {
             throw new CreateException(
               "Name cannot be null or empty.");
           }
           else {
             customerHelper = new
               CustomerHelper(person, password, email);
```

```
            customerHelper.setName(person);
            customerHelper.setEmail(email);
            customerHelper.setPassword(password);
          }
          shoppingList = new ArrayList();
}

 public void ejbCreate(CustomerHelper customerHelper)
    throws CreateException {
        customerHelper.setName(customerHelper.name);
        customerHelper.setEmail(customerHelper.email);
        customerHelper.setPassword(customerHelper.password);
        shoppingList = new ArrayList();
}
 // Business methods implementation

public CustomerHelper getCustomer() {
    return customerHelper;
}
```

Business object sample

The following sample screen illustrates the business object structure that corresponds to the source code presented in "EJB JAR file code" on page 34. This business object is created by the ODA, which discovers the objects and constructs defined in the original EJB JAR file and generates corresponding business objects. For information about how to use the ODA to generate this example, see Chapter 5, "Creating and modifying business objects," on page 37.

Notice the fourth attribute, `getCustomer`. This attribute contains a complex method object that corresponds to the `getCustomer` method defined at the end of the sample code for the `MusicCartBean` class, presented in "EJB JAR file code" on page 34.



	Name	Type	Key	Foreign Key	Required Attribute	Cardinality	Maximum	Default	Application Specific Information
1	⊞ getAllRecordInfo	MusicCart_getAllRecordInfo	☐	☐	☐	1			method_name=getAllRecordInfo
2	⊞ modifyMusicRequestUsingDataHandler	MusicCart_modifyMusicRequestUsingDataHandler	☐	☐	☐	1			method_name=modifyMusicRequestUsingDataHandler
3	⊞ getFirstRecordInfo	MusicCart_getFirstRecordInfo	☐	☐	☐	1			method_name=getFirstRecordInfo
4	⊟ getCustomer	MusicCart_getCustomer	☐	☐	☐	1			method_name=getCustomer
4.1	⊟ Return_Value	CustomerHelper	☐	☐	☐	1			type=proxy
4.1.1	name	String	☐	☐	☐		255		property=name;type=String
4.1.2	password	String	☐	☐	☐		255		property=password;type=String
4.1.3	email	String	☐	☐	☐		255		property=email;type=String
4.1.4	⊞ CustomerHelper_0	CustomerHelper_0	☐	☐	☐	1			method_name=CONSTRUCTOR
4.1.5	⊞ setEmail	CustomerHelper_setEmail	☐	☐	☐	1			method_name=setEmail
4.1.6	⊞ setName	CustomerHelper_setName	☐	☐	☐	1			method_name=setName
4.1.7	⊞ setPassword	CustomerHelper_setPassword	☐	☐	☐	1			method_name=setPassword
4.1.8	⊞ getEmail	CustomerHelper_getEmail	☐	☐	☐	1			method_name=getEmail
4.1.9	⊞ getName	CustomerHelper_getName	☐	☐	☐	1			method_name=getName
4.1.10	⊞ getPassword	CustomerHelper_getPassword	☐	☐	☐	1			method_name=getPassword

Figure 4. Business object level ASI and supported verbs

Generating business objects

Each time an event occurs during run time, an EJB application sends a message object containing object-level data and information about the type of transaction. The connector maps this data to the corresponding business object definition, to create an application-specific business object. The connector sends these business

objects on to the integration broker for processing. It also receives business objects back from the integration broker, which it passes back to the EJB application.

Note: If the object model in the EJB application is changed, use the ODA to create a new definition. If the business object definitions in the integration broker repository do not match exactly the data that the EJB application sends, the connector is unable to create a business object and the transaction will fail.

Business Object Designer provides a graphical interface that enables you to create and modify business object definitions for use at run time. For details, see Chapter 5, “Creating and modifying business objects,” on page 37.

Chapter 5. Creating and modifying business objects

This chapter describes how to create business objects using the ODA (Object Discovery Agent) for EJB, and contains the following sections:

Overview of the ODA for EJB

An ODA enables you to generate business object definitions. A business object definition is a template for a business object. The ODA examines specified application objects, "discovers" the elements of those objects that correspond to business object attributes, and generates business object definitions to represent the information. Business Object Designer provides a graphical interface to access the Object Discovery Agent and to work with it interactively.

The ODA for EJB generates business object definitions from metadata contained in EJB JAR files. The Business Object Designer wizard automates the process of creating these definitions. You use the ODA to create business objects and Connector Configurator to configure the connector to support them. For information about Connector Configurator, see Appendix B, "Connector Configurator," on page 67.

Generating business object definitions

This section describes how to use the EJB ODA in Business Object Designer to generate business object definitions. For information on launching and using Business Object Designer, see *IBM WebSphere Business Integration Adapters Business Object Development Guide*.

Starting the ODA

The ODA can be run from any machine that can mount the file system on which the metadata repository (the JAR files) resides, using the ODA startup file. This file contains start parameters, including the paths to certain required EJB and connector JAR files. These JAR files must also be accessible from the machine on which you are running the ODA.

The ODA for EJB has a default name of EJBODA. The name can be changed by changing the value of the `AGENTNAME` variable in the start script.

To start the ODA, run this command:

- `start_EJBODA.bat` (Windows 2000)
- `start_EJBODA.sh` (Unix)

The `CLIENT JARPATH` variable of the startup file contains no default value, so before running the `start` command, set the value to the directory and filename of the client JAR path, so as to allow the ODA to include the file in its Java class path. For example:

```
set CLIENTJARPATH=C:\BIA_MusicBeanSample.jar
```

© Copyright IBM Corp. 2003, 2004

Running Business Object Designer

Business Object Designer provides a wizard that guides you through the steps to generate a business object definition using the ODA. The steps are as follows:

Select the agent

To select the agent, follow these steps.

1. Start Business Object Designer.
2. Click **File > New Using ODA**. The *Business Object Wizard - Step 1 of 6 - Select Agent* screen appears.
3. Select the ODA/AGENTNAME in the **Located agents** list and click **Next**. (You may have to click **Find Agents** if the desired agent is not listed.)

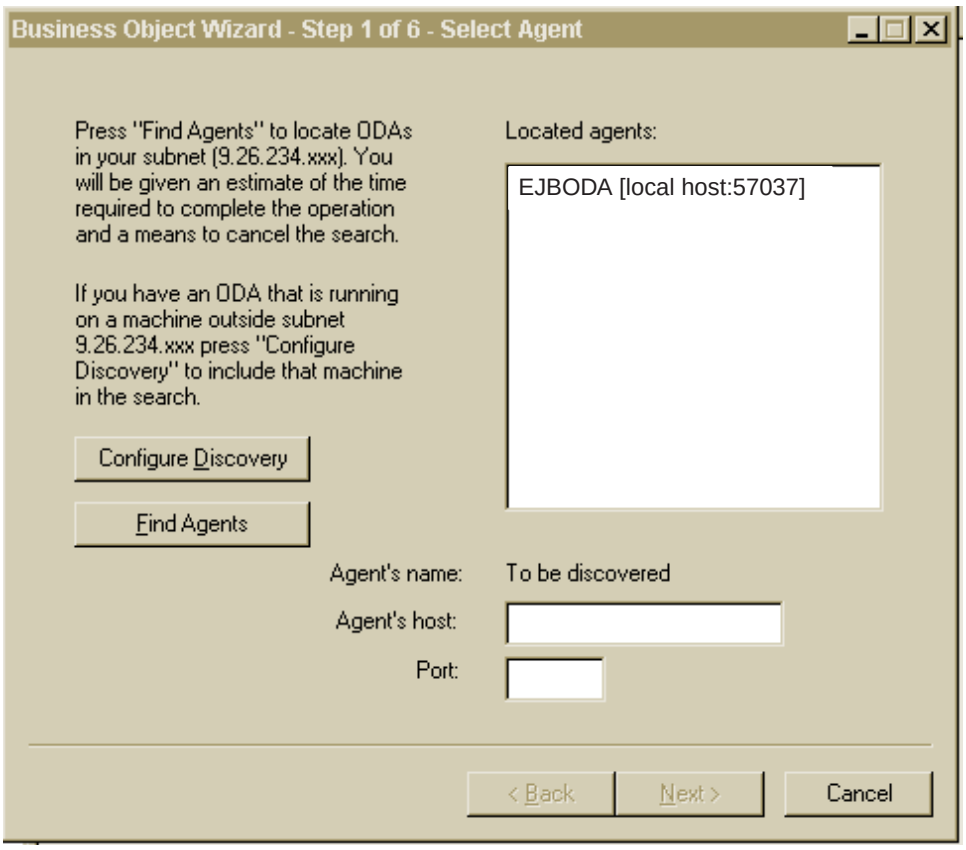


Figure 5. Select Agent screen

Configure the agent

After you click **Next** on the Select Agent screen, the *Business Object Wizard - Step 2 of 6 - Configure Agent* screen appears. Figure 6 on page 39 illustrates this screen with sample values.

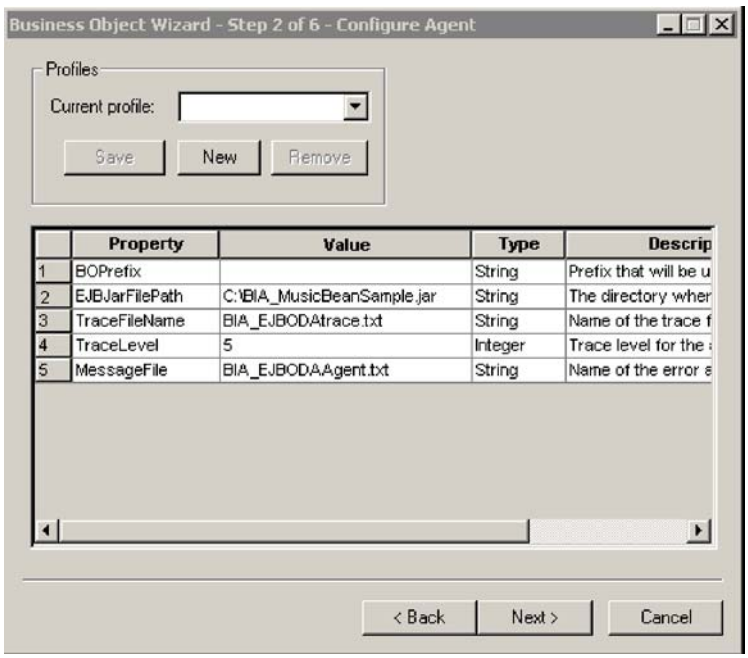


Figure 6. Configure Agent screen

The properties you set on this screen are described in Table 9. You can save all the values you enter on this screen to a profile. Instead of retyping the property data the next time you run the ODA, you simply select a profile from the Current profile drop-down menu and re-use the saved values. You can save multiple profiles, each with a different set of specified values.

Table 9. Configure Agent properties

Property name	Default value	Type	Description
`EJBJarFilePath`	None	String	(required) The fully qualified path to the EJB JAR file. This is the file that contains the definitions of the enterprise beans with which the BO must exchange data.
`DefaultBOPrefix`	None	String	The prefix that the ODA will add to the names of the business objects it generates
`TraceFileName`	*<agentname>*`trace.txt`	String	The name of the trace message file; for example, `BIA_EJBODAtrace.txt`. Note that the connector supports globalized values for this property, which means that the file path and name can be globalized, though the file content cannot.
`TraceLevel`	5	Integer	(required) The tracing level (from 0 to 5) for the Agent. For details about tracing levels, see “Tracing” on page 48.
`MessageFile`	*<agentname>*`Agent.txt`	String	The name of the message file that contains all the messages displayed by the ODA. For the connector for EJB, the name of this file is `BIA_EJBODAAgent.txt`. If you do not correctly specify the name of the message file, the ODA will run without messages. Note that the connector supports globalized values for this property, which means that the file path and name can be globalized, though the file content cannot.

1. Use the **New** and **Save** buttons in the Profiles group box the first time you run the ODA to create a new profile. When you use the ODA again, you can select an existing profile.
2. Type the name of each property, its value, type, and description, as defined in Table 9 on page 39.

   Note: If you use a profile, the property values are filled in for you, though you can modify the values as needed.

Select a business object

The *Business Object Wizard - Step 3 of 6 - Select Source* screen appears, as illustrated in Figure 7. The screen lists the objects that have been defined in the EJB JAR file. Use this screen to select any number of EJB objects for which the ODA will generate business object definitions.

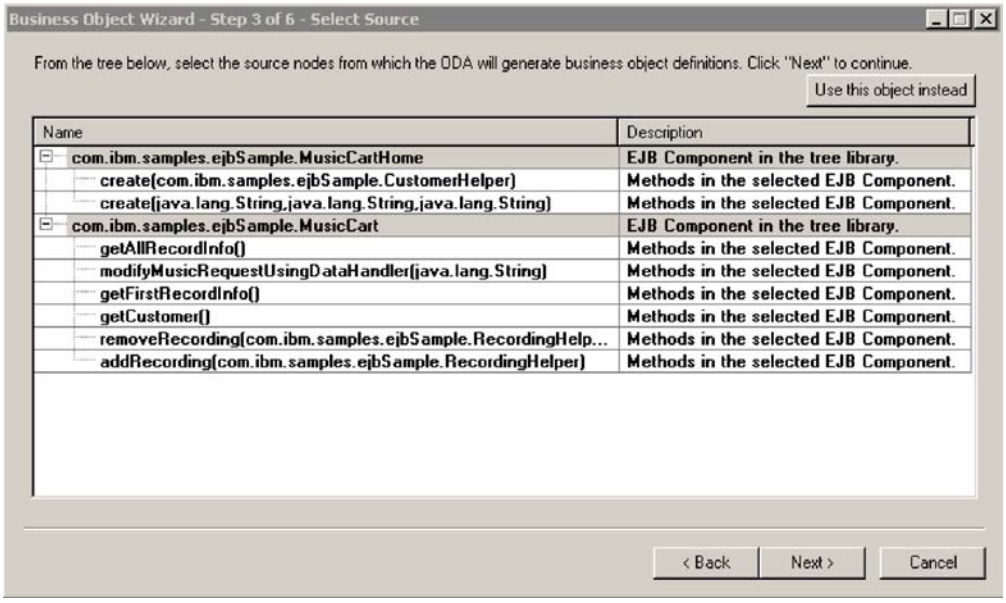


Figure 7. Select Source screen

1. If necessary, expand an EJB object to see a list of methods of the object.
2. Select the EJB object(s) you want to use. In Figure 7, the `MusicCartHome` and `MusicCart` class are selected. Although only the class names are selected, the ODA will also create objects of the associated methods of each selected class.
3. Click **Next**.

Confirm the object selection

The *Business Object Wizard - Step 4 of 6 - Confirm source nodes for business object definitions* screen appears. It shows the object(s) you selected.

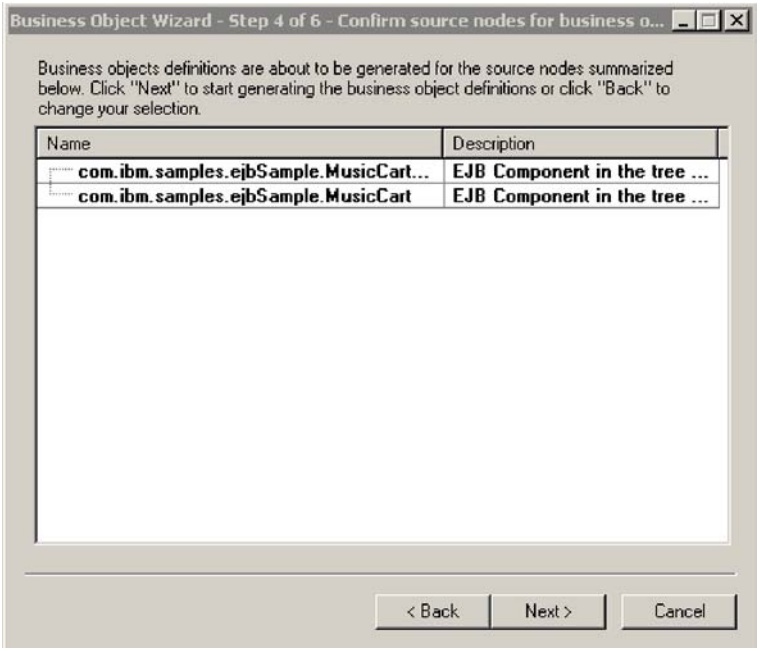


Figure 8. Confirm source node screen

Click **Back** to make changes or **Next** to confirm that the list is correct.

The *Business Object Wizard - Step 5 of 6 - Generating business objects...* screen appears with a message stating that the wizard is generating the business objects.

Select verbs and assign the JNDI name

After you click **Next**, the *Enter JNDI name for this Enterprise Java Bean* screen appears. Use this screen to select the supported verbs for this object and to assign a JNDI name. In the example provided in Figure 9 on page 42, the JNDI name assigned is `MusicCartJNDI`.

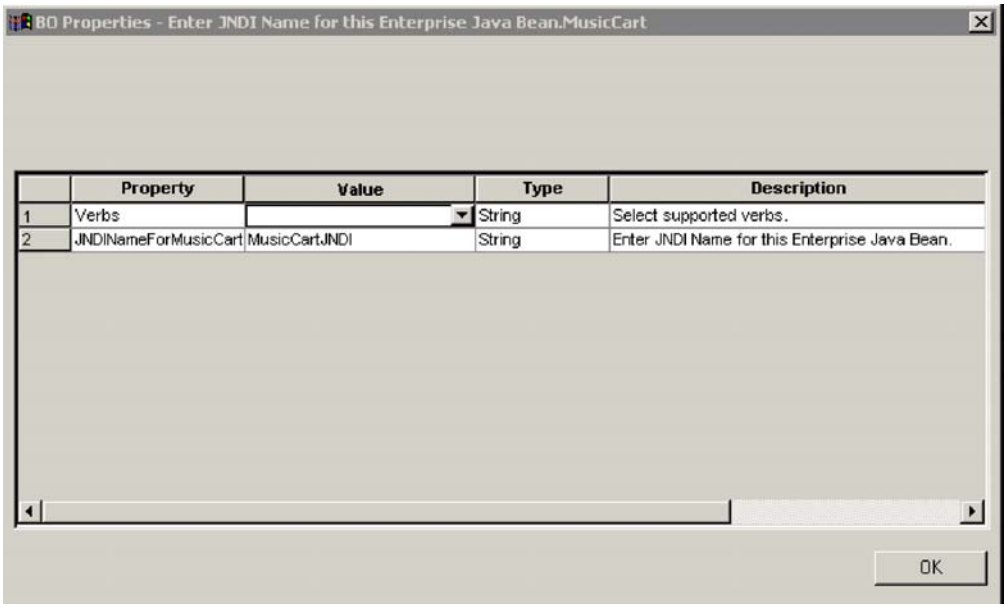
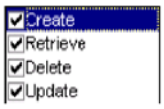


Figure 9. Enter JNDI Name for Enterprise Java Bean screen

On this screen you also specify the verbs that the business object supports. The ODA allows you to specify the four supported verbs (Create, Retrieve, Delete, and Update) and assign as actions of each verb `n` methods, where `n` equals the number of methods in the corresponding EJB object. To specify additional verbs beyond the supported four, or to edit verb information after you create a business object, use Business Object Designer.

For details about business object verbs for the EJB connector, see "Verb ASI" on page 28.

1. In the **Value** list for the `Verbs` property, select the verbs that you want the business object to support. You can select one or more verbs. You can also deselect a verb at any time.



2. In the `JNDI name for <className>` property, enter the JNDI name of the enterprise bean. This name is defined at the time when the bean is deployed onto the application server.
3. Click **OK**.

Specify the verb ASI

For each verb you select, a separate window appears where you specify the method sequence that must be executed for the verb.

Figure 10 illustrates this screen for the `Create` verb of the `MusicCart` business object created in Figure 7 on page 40 and Figure 8 on page 41.

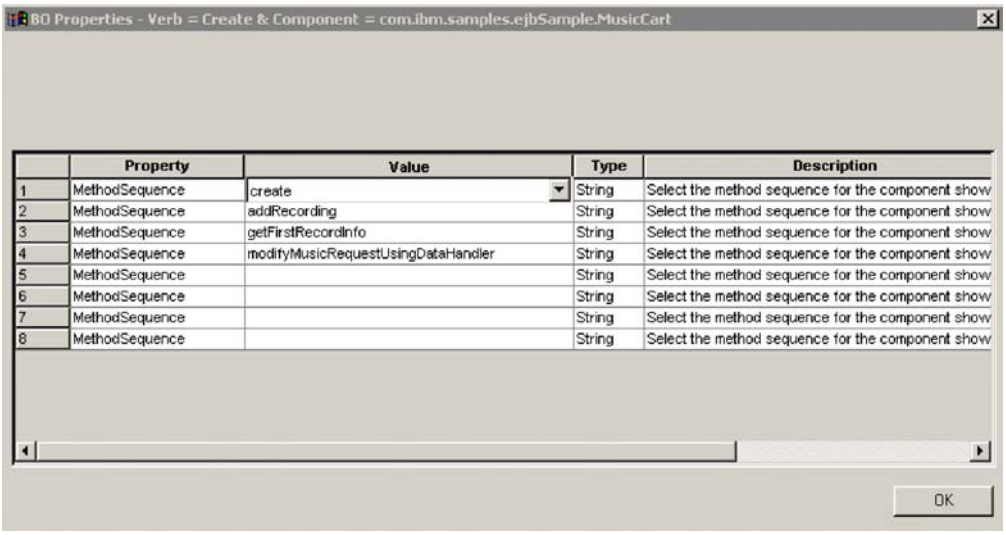


Figure 10. Setting the verb method sequence

1. In the **Value** list for the `MethodSequence` property, click the method that you want the business object to execute first for the verb. In Figure 10 on page 42, the method sequence is as follows:
   - The first method required in the method sequence of any EJB business object verb is a creator or finder method, defined in the home interface of the enterprise bean. In Figure 10 on page 42 the first method is `create`, which the connector uses to obtain a reference to the remote interface and create an instance of the enterprise bean.
   - The second method in the sequence is `addRecording`. This method belongs to the enterprise bean's remote interface.
   - The third method in the sequence is `getFirstRecordInfo`. This method belongs to the enterprise bean's remote interface.
   - The last method in the sequence is `modifyMusicRequestUsingDataHandler`. This method belongs to the enterprise bean's remote interface.

   By specifying a method sequence for the verb, you are creating the verb ASI that is associated with that verb. If necessary, this verb ASI can be modified later using Business Object Designer.
2. Click **OK**.

Opening or saving the business object

The *Business Object Wizard - Step 6 of 6 - Save business objects* screen appears.

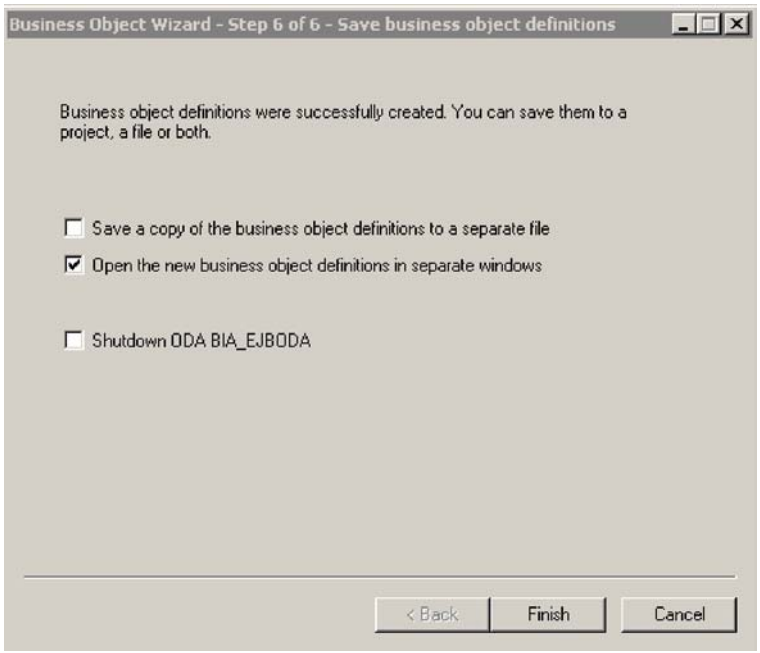


Figure 11. Save business objects screen

You can optionally open the new business objects in separate windows within Business Object Designer, or (after specifying a key for the top-level business object) you can save the generated business object definitions to a file.

To open the business objects in separate windows

1. Select **Open the new BOs in separate windows**.

2. Click **Finish**. Each business object appears in a separate window where you can view and set the ASI information for the business objects and business object verbs you just created. For details, see "Select verbs and assign the JNDI name" on page 41 and "Specify the verb ASI" on page 42.

To save the business objects to a file (only after you specify a key for the parent-level business object, as illustrated in Figure 12 on page 45):

1. Select **Save a copy of the business objects to a separate file**. A dialog box appears.
2. Type the location in which you want the copy of the new business object definitions to be saved.

Business Object Designer saves the files to the specified location.

If you have finished working with the ODA, you can shut it down by checking "Shutdown ODA EJBODA" before clicking **Finish**.

Specifying the attribute-level ASI

After you define the verb ASI (by specifying a method sequence that must be executed for each verb), Business Object Designer displays the attributes for the business object. For details about the attribute-level ASI in the EJB connector, see "Attribute-level ASI" on page 29.

The attributes are listed on the **Attributes** tab in the order in which they appear in the business object structure, as defined by the numeric value in the **Pos** column. Simple EJB object attributes are represented as simple attributes and their ASI contains the original EJB attribute name and type.

For each attribute, the screen provides the name of the attribute, its type, and the ASI information. Figure 12 on page 45 illustrates the method (complex) attribute ASI. The `getAllRecordsInfo` attribute of the business object has an ASI that maps the attribute to the original EJB object method. In this example, the original method is indicated under the **App Spec Info** column, by the `method_name=getAllRecordInfo` ASI.

The `getCustomer` attribute of the business object has an ASI that maps the attribute to the original EJB object method. In this example, the original method is indicated under the **App Spec Info** column, by the `method_name=getCustomer` ASI.

In addition, `getCustomer` (a child business object) has the child object attribute `Return_Value`, which is used to capture the return value of the `getCustomer` method. In the EJB JAR file, the method is defined as having a return value of type `CustomerHelper`, which itself is a method. The attribute ASI of `CustomerHelper` is set to `type=proxy` because the object type is a return value. Note that if a method in the EJB JAR file does not return a value, the `Return_Value` attribute is not included in the list of business object attributes.

Notice also that the `Return_Value` attribute of type `CustomerHelper` has three non-method child object attributes: `name`, `password`, and `email`. In the original EJB JAR file where the `MusicCartBean` class is defined, these are all defined as parameters of type `String` of the `CustomerHelper` method. In the business object, the attribute ASI of these parameters indicates the Java property name and type (`String`) of each in the original JAR file.

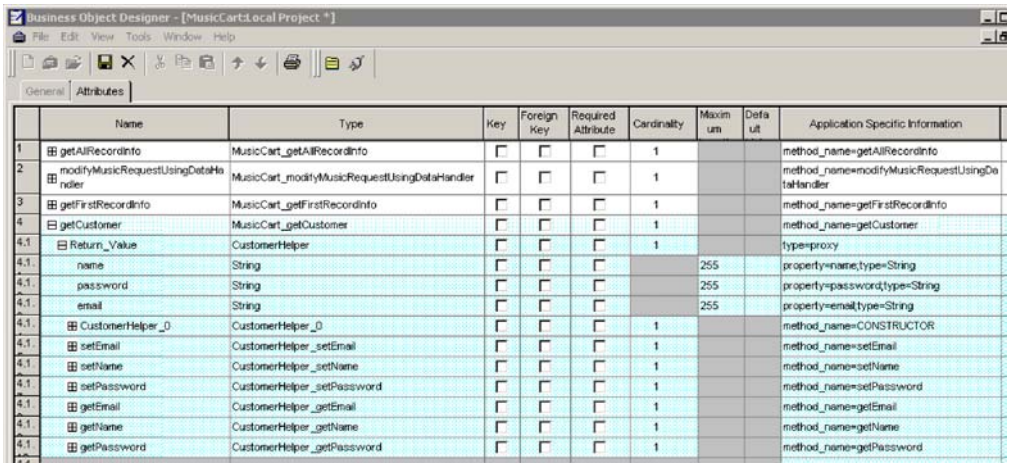


Figure 12. Setting the attribute ASI

On this screen, you should specify whether or not a parent-level object is a key (which is required by the ODA for saving the business objects to a separate file). You can also use this screen to set child object keys as needed and to specify the following information:

- Is the attribute required for the connector to process the business object? If so, click the **Required** check box.
- Is the maximum length of the attribute different from the value that appears in the **Maximum Length** column?
- Does the attribute have a default value? If so, type the value in the **Default** column.

Note: While you can create a business object through the ODA (running in Business Object Designer) and set the parent level key(s), do not configure the foreign key in this manner. The foreign key is non-ASI metadata and therefore must always be configured without the ODA (in Business Object Designer, click **File** > **New** to create a new business object without using the ODA).

Specifying the business object-level ASI

After specifying the attribute-level ASI, you can view and modify the business object-level ASI. For details about business object-level ASI, see "Business object-level ASI" on page 27.

The business object-level ASI is listed on the **General** tab. The ASI value that appears in the field **Business Object Level Application-specific information** contains the name of the remote interface that corresponds to this business object. The connector uses this information to map a remote object to a business object.

The **General** tab also lists all the verbs that are supported by the business object and provides the ASI for each verb, as it was defined in "Specify the verb ASI" on page 42. If a verb is blank, then a method sequence will not be executed for that verb.

Uploading business object files

The newly created business object definition files must be uploaded to the integration broker once they have been created. The process depends on whether you are running WebSphere InterChange Server, WebSphere MQ Integrator Broker, or WebSphere Application Server.

- **WebSphere InterChange Server**: If you have saved your business object definition files to a local machine and must upload them to the repository on the server, refer to the InterChange Server implementation documentation.
- **WebSphere MQ Integrator Broker**: You must export the business object definitions out of Business Object Designer and into the integration broker. For details, refer to the implementation documentation of WebSphere MQ Integrator Broker.
- **WebSphere Application Server**: For details, see the implementation documentation of WebSphere Application Server.

Chapter 6. Troubleshooting and error handling

This chapter describes how the connector for EJB handles errors. The connector generates logging and tracing messages. This chapter describes these messages and provides troubleshooting tips. The chapter contains the following sections:

- "Error handling"
- "Logging" on page 48
- "Tracing" on page 48

Error handling

All messages generated by the connector are stored in a message file named `BIA_EJBConnector.txt`. (The name of the file is determined by the `LogFileName` standard connector configuration property.) Each message has a message number followed by the message:

```
Message number
Message text
```

The connector handles specific errors as described in the following sections.

ClassNotFound for proxy

An exception is raised when the connector cannot find a given remote interface in the EJB JAR file provided by the user for exchanging data with the connector. The connector logs the error, which includes the name of the class not found, and returns a FAIL code.

InstantiationException in Loader

When the connector receives the enterprise bean class name and tries to create an object of that class, an exception is raised if it cannot create the object instance. The connector logs the error, which includes the class name of the object that cannot be instantiated, and returns a FAIL code.

Illegal AccessException in Loader or Invoker

The connector raises an exception due to invalid code or improper access (public or private) on a method.

The connector logs the error and returns a FAIL code.

NoSuchMethodException in Invoker

The connector raises an exception if a method is specified on the business object that does not exist in the corresponding enterprise bean object. Since methods are loaded dynamically, this exception is raised when the method is not found in the class.

The connector logs the error and returns a FAIL code.

InvocationTargetException in Invoker

The connector raises an exception when the EJB application (with which the connector is exchanging business objects) raises an exception.

© Copyright IBM Corp. 2003, 2004

The connector logs the error and returns a FAIL code.

Invalid argument (CXIgnore) in a method object in Invoker

The connector raises an exception when a method is included in the business object's verb ASI, but the arguments of that method have not been populated.

The connector logs the error and returns a FAIL code.

Cast failure or wrong attribute type

The connector raises an exception if an EJB object method takes or returns a different data type than what has been specified in the business object.

The connector logs the error and returns a FAIL code.

Invalid verb ASI

The connector raises an exception if the verb ASI of the business object being passed to it is formatted incorrectly or uses improper syntax. Examples of this include a verb ASI that does not contain a proper method sequence.

The connector logs the error and returns a FAIL code.

App response timeout

The connector raises an exception if it loses the connection to the Application Server where the enterprise beans are deployed or if it cannot invoke an EJB method due to a connection failure.

The connector logs the error and returns a FAIL code.

Logging

All errors described in "Error handling" on page 47 must be read from the message file (`BIA_EJBConnector.txt`).

Tracing

Tracing is an optional debugging feature you can turn on to closely follow connector behavior. Trace messages, by default, are written to `STDOUT`. For more on configuring trace messages, see the connector configuration properties in "Configuring the connector" on page 13. For more information on tracing, including how to enable and set it, see the *Connector Development Guide*.

Table 10 lists the recommended content for connector tracing message levels.

Table 10. Tracing messages content

Level	Description
Level 0	Use this level for trace messages that identify the connector version. No other tracing is performed at this level.
Level 1	Use this level for trace messages that: • Provide status information. • Provide key information on each business object processed. • Record each time a polling thread detects a new message in an input queue.

Table 10. Tracing messages content (continued)

Level	Description
Level 2	Use this level for trace messages that: • Identify the BO handler used for each object that the connector processes. • Log each time a business object is posted to the integration broker. • Indicate each time a request business object is received.
Level 3	Use this level for trace messages that: • Identify the foreign keys being processed, if applicable. These messages appear when the connector has encountered a foreign key in a business object or when the connector sets a foreign key in a business object. • Relate to business object processing. Examples of this include finding a match between business objects, or finding a business object in an array of child business objects.
Level 4	Use this level for trace messages that: • Identify application-specific information. Examples of this include the values returned by the methods that process the application-specific information fields in business objects. • Identify when the connector enters or exits a function. These messages help trace the process flow of the connector. • Record any thread-specific processing. For example, if the connector spawns multiple threads, a message logs the creation of each new thread.
Level 5	Use this level for trace messages that: • Indicate connector initialization. This type of message can include, for example, the value of each connector configurator property that has been retrieved from the broker. • Detail the status of each thread that the connector spawns while it is running. • Represent statements executed in the application. The connector log file contains all statements executed in the target application and the value of any variables that are substituted, where applicable. • Record business object dumps. The connector should output a text representation of a business object before it begins processing (showing the object that the connector receives from the collaboration) as well as after it finishes processing the object (showing the object that the connector returns to the collaboration).

Appendix A. Standard configuration properties for connectors

This appendix describes the standard configuration properties for the connector component of WebSphere Business Integration adapters. The information covers connectors running on the following integration brokers:

- WebSphere InterChange Server (ICS)
- WebSphere MQ Integrator, WebSphere MQ Integrator Broker, and WebSphere Business Integration Message Broker, collectively referred to as the WebSphere Message Brokers (WMQI).
- WebSphere Application Server (WAS)

Not every connector makes use of all these standard properties. When you select an integration broker from Connector Configurator, you will see a list of the standard properties that you need to configure for your adapter running with that broker.

For information about properties specific to the connector, see the relevant adapter user guide.

Note: In this document, backslashes (\) are used as the convention for directory paths. For UNIX installations, substitute slashes (/) for backslashes and follow the conventions for each operating system.

New and deleted properties

These standard properties have been added in this release.

New properties

- XMLNameSpaceFormat

Deleted properties

- RestartCount

Configuring standard connector properties

Adapter connectors have two types of configuration properties:

- Standard configuration properties
- Connector-specific configuration properties

This section describes the standard configuration properties. For information on configuration properties specific to a connector, see its adapter user guide.

Using Connector Configurator

You configure connector properties from Connector Configurator, which you access from System Manager. For more information on using Connector Configurator, refer to the Connector Configurator appendix.

Note: Connector Configurator and System Manager run only on the Windows system. If you are running the connector on a UNIX system, you must have a Windows machine with these tools installed. To set connector properties

© Copyright IBM Corp. 2003, 2004

for a connector that runs on UNIX, you must start up System Manager on the Windows machine, connect to the UNIX integration broker, and bring up Connector Configurator for the connector.

Setting and updating property values

The default length of a property field is 255 characters.

The connector uses the following order to determine a property's value (where the highest number overrides other values):

1. Default
2. Repository (only if WebSphere InterChange Server is the integration broker)
3. Local configuration file
4. Command line

A connector obtains its configuration values at startup. If you change the value of one or more connector properties during a run-time session, the property's **Update Method** determines how the change takes effect. There are four different update methods for standard connector properties:

- **Dynamic**
  The change takes effect immediately after it is saved in System Manager. If the connector is working in stand-alone mode (independently of System Manager), for example with one of the WebSphere message brokers, you can only change properties through the configuration file. In this case, a dynamic update is not possible.
- Component restart
  The change takes effect only after the connector is stopped and then restarted in System Manager. You do not need to stop and restart the application-specific component or the integration broker.
- Server restart
  The change takes effect only after you stop and restart the application-specific component and the integration broker.
- Agent restart (ICS only)
  The change takes effect only after you stop and restart the application-specific component.

To determine how a specific property is updated, refer to the **Update Method** column in the Connector Configurator window, or see the Update Method column in the Property Summary table below.

Summary of standard properties

Table 11 on page 53 provides a quick reference to the standard connector configuration properties. Not all the connectors make use of all these properties, and property settings may differ from integration broker to integration broker, as standard property dependencies are based on `RepositoryDirectory`.

You must set the values of some of these properties before running the connector. See the following section for an explanation of each property.

Table 11. Summary of standard configuration properties

Property name	Possible values	Default value	Update method	Notes
AdminInQueue	Valid JMS queue name	*CONNECTORNAME* /ADMININQUEUE	Component restart	Delivery Transport is JMS
AdminOutQueue	Valid JMS queue name	*CONNECTORNAME*/ADMINOUTQUEUE	Component restart	Delivery Transport is JMS
AgentConnections	1-4	1	Component restart	Delivery Transport is MQ or IDL: Repository directory is <REMOTE>
AgentTraceLevel	0-5	0	Dynamic	
ApplicationName	Application name	Value specified for the connector application name	Component restart	
BrokerType	ICS, WMQI, WAS			
CharacterEncoding	ascii7, ascii8, SJIS, Cp949, GBK, Big5, Cp297, Cp273, Cp280, Cp284, Cp037, Cp437 **Note:** This is a subset of supported values.	ascii7	Component restart	
ConcurrentEventTriggeredFlows	1 to 32,767	1	Component restart	Repository directory is <REMOTE>
ContainerManagedEvents	No value or JMS	No value	Component restart	Delivery Transport is JMS
ControllerStoreAndForwardMode	true or false	True	Dynamic	Repository directory is <REMOTE>
ControllerTraceLevel	0-5	0	Dynamic	Repository directory is <REMOTE>
DeliveryQueue		*CONNECTORNAME*/DELIVERYQUEUE	Component restart	JMS transport only
DeliveryTransport	MQ, IDL, or JMS	JMS	Component restart	If Repository directory is local, then value is JMS only
DuplicateEventElimination	True or False	False	Component restart	JMS transport only: Container Managed Events must be <NONE>
FaultQueue		*CONNECTORNAME*/FAULTQUEUE	Component restart	JMS transport only

Table 11. Summary of standard configuration properties (continued)

Property name	Possible values	Default value	Update method	Notes
jms.FactoryClassName	CxCommon.Messaging.jms .IBMMQSeriesFactory or CxCommon.Messaging .jms.SonicMQFactory or any Java class name	CxCommon.Messaging. jms.IBMMQSeriesFactory	Component restart	JMS transport only
jms.MessageBrokerName	If FactoryClassName is IBM, use crossworlds.queue. manager. If FactoryClassName is Sonic, use localhost:2506.	crossworlds.queue.manager	Component restart	JMS transport only
jms.NumConcurrentRequests	Positive integer	10	Component restart	JMS transport only
jms.Password	Any valid password		Component restart	JMS transport only
jms.UserName	Any valid name		Component restart	JMS transport only
JvmMaxHeapSize	Heap size in megabytes	128m	Component restart	Repository directory is <REMOTE>
JvmMaxNativeStackSize	Size of stack in kilobytes	128k	Component restart	Repository directory is <REMOTE>
JvmMinHeapSize	Heap size in megabytes	1m	Component restart	Repository directory is <REMOTE>
ListenerConcurrency	1- 100	1	Component restart	Delivery Transport must be MQ
Locale	en_US, ja_JP, ko_KR, zh_CN, zh_TW, fr_FR, de_DE, it_IT, es_ES, pt_BR **Note:** This is a subset of the supported locales.	en_US	Component restart	
LogAtInterchangeEnd	True or False	False	Component restart	Repository Directory must be <REMOTE>
MaxEventCapacity	1-2147483647	2147483647	Dynamic	Repository Directory must be <REMOTE>
MessageFileName	Path or filename	InterchangeSystem.txt	Component restart	
MonitorQueue	Any valid queue name	*CONNECTORNAME*/MONITORQUEUE	Component restart	JMS transport only: DuplicateEvent Elimination must be True
OADAutoRestartAgent	True or False	False	Dynamic	Repository Directory must be <REMOTE>

Table 11. Summary of standard configuration properties (continued)

Property name	Possible values	Default value	Update method	Notes
OADMaxNumRetry	A positive number	1000	Dynamic	Repository Directory must be <REMOTE>
OADRetryTimeInterval	A positive number in minutes	10	Dynamic	Repository Directory must be <REMOTE>
PollEndTime	HH:MM	HH:MM	Component restart	
PollFrequency	A positive integer in milliseconds no (to disable polling) key (to poll only when the letter p is entered in the connector's Command Prompt window)	10000	Dynamic	
PollQuantity	1-500	1	Agent restart	JMS transport only: Container Managed Events is specified
PollStartTime	HH:MM(HH is 0-23, MM is 0-59)	HH:MM	Component restart	
RepositoryDirectory	Location of metadata repository		Agent restart	For ICS: set to <REMOTE> For WebSphere MQ message brokers and WAS: set to C:\crossworlds\ repository
RequestQueue	Valid JMS queue name	*CONNECTORNAME*/REQUESTQUEUE	Component restart	Delivery Transport is JMS
ResponseQueue	Valid JMS queue name	*CONNECTORNAME*/RESPONSEQUEUE	Component restart	Delivery Transport is JMS: required only if Repository directory is <REMOTE>
RestartRetryCount	0-99	3	Dynamic	
RestartRetryInterval	A sensible positive value in minutes: 1 - 2147483547	1	Dynamic	
RHF2MessageDomain	mrm, xml	mrm	Component restart	Only if Delivery Transport is JMS and WireFormat is CwXML.

Table 11. Summary of standard configuration properties (continued)

Property name	Possible values	Default value	Update method	Notes
SourceQueue	Valid WebSphere MQ name	*CONNECTORNAME*/SOURCEQUEUE	Agent restart	Only if Delivery Transport is JMS and Container Managed Events is specified
SynchronousRequestQueue		*CONNECTORNAME*/ SYNCHRONOUSREQUESTQUEUE	Component restart	Delivery Transport is JMS
SynchronousRequestTimeout	0 - any number (millisecs)	0	Component restart	Delivery Transport is JMS
SynchronousResponseQueue		*CONNECTORNAME*/ SYNCHRONOUSRESPONSEQUEUE	Component restart	Delivery Transport is JMS
WireFormat	CwXML, CwBO	CwXML	Agent restart	CwXML if Repository Directory is not <REMOTE>: CwBO if Repository Directory is <REMOTE>
WsifSynchronousRequest Timeout	0 - any number (millisecs)	0	Component restart	WAS only
XMLNameSpaceFormat	short, long	short	Agent restart	WebSphere MQ message brokers and WAS only

Standard configuration properties

This section lists and defines each of the standard connector configuration properties.

AdminInQueue

The queue that is used by the integration broker to send administrative messages to the connector.

The default value is CONNECTORNAME/ADMININQUEUE.

AdminOutQueue

The queue that is used by the connector to send administrative messages to the integration broker.

The default value is CONNECTORNAME/ADMINOUTQUEUE.

AgentConnections

Applicable only if RepositoryDirectory is <REMOTE>.

The AgentConnections property controls the number of ORB connections opened by orb.init[].

By default, the value of this property is set to 1. There is no need to change this default.

AgentTraceLevel

Level of trace messages for the application-specific component. The default is `0`. The connector delivers all trace messages applicable at the tracing level set or lower.

ApplicationName

Name that uniquely identifies the connector's application. This name is used by the system administrator to monitor the WebSphere business integration system environment. This property must have a value before you can run the connector.

BrokerType

Identifies the integration broker type that you are using. The options are ICS, WebSphere message brokers (WMQI, WMQIB or WBIMB) or WAS.

CharacterEncoding

Specifies the character code set used to map from a character (such as a letter of the alphabet, a numeric representation, or a punctuation mark) to a numeric value.

Note: Java-based connectors do not use this property. A C++ connector currently uses the value `ascii7` for this property.

By default, a subset of supported character encodings only is displayed in the drop list. To add other supported values to the drop list, you must manually modify the `\Data\Std\stdConnProps.xml` file in the product directory. For more information, see the appendix on Connector Configurator.

ConcurrentEventTriggeredFlows

Applicable only if `RepositoryDirectory` is <REMOTE>.

Determines how many business objects can be concurrently processed by the connector for event delivery. Set the value of this attribute to the number of business objects you want concurrently mapped and delivered. For example, set the value of this property to `5` to cause five business objects to be concurrently processed. The default value is `1`.

Setting this property to a value greater than `1` allows a connector for a source application to map multiple event business objects at the same time and deliver them to multiple collaboration instances simultaneously. This speeds delivery of business objects to the integration broker, particularly if the business objects use complex maps. Increasing the arrival rate of business objects to collaborations can improve overall performance in the system.

To implement concurrent processing for an entire flow (from a source application to a destination application), you must:

- Configure the collaboration to use multiple threads by setting its `Maximum number of concurrent events` property high enough to use multiple threads.
- Ensure that the destination application's application-specific component can process requests concurrently. That is, it must be multi-threaded, or be able to use connector agent parallelism and be configured for multiple processes. Set the `Parallel Process Degree` configuration property to a value greater than `1`.

The ConcurrentEventTriggeredFlows property has no effect on connector polling, which is single-threaded and performed serially.

ContainerManagedEvents

This property allows a JMS-enabled connector with a JMS event store to provide guaranteed event delivery, in which an event is removed from the source queue and placed on the destination queue as a single JMS transaction.

The default value is No value.

When ContainerManagedEvents is set to JMS, you must configure the following properties to enable guaranteed event delivery:

- PollQuantity = 1 to 500
- SourceQueue = CONNECTORNAME/SOURCEQUEUE

You must also configure a data handler with the MimeType, DHClass, and DataHandlerConfigMOName (optional) properties. To set those values, use the **Data Handler** tab in Connector Configurator. The fields for the values under the Data Handler tab will be displayed only if you have set ContainerManagedEvents to JMS.

Note: When ContainerManagedEvents is set to JMS, the connector does *not* call its pollForEvents() method, thereby disabling that method's functionality.

This property only appears if the DeliveryTransport property is set to the value JMS.

ControllerStoreAndForwardMode

Applicable only if RepositoryDirectory is <REMOTE>.

Sets the behavior of the connector controller after it detects that the destination application-specific component is unavailable.

If this property is set to true and the destination application-specific component is unavailable when an event reaches ICS, the connector controller blocks the request to the application-specific component. When the application-specific component becomes operational, the controller forwards the request to it.

However, if the destination application's application-specific component becomes unavailable **after** the connector controller forwards a service call request to it, the connector controller fails the request.

If this property is set to false, the connector controller begins failing all service call requests as soon as it detects that the destination application-specific component is unavailable.

The default is true.

ControllerTraceLevel

Applicable only if RepositoryDirectory is <REMOTE>.

Level of trace messages for the connector controller. The default is 0.

DeliveryQueue

Applicable only if DeliveryTransport is JMS.

The queue that is used by the connector to send business objects to the integration broker.

The default value is CONNECTORNAME/DELIVERYQUEUE.

DeliveryTransport

Specifies the transport mechanism for the delivery of events. Possible values are MQ for WebSphere MQ, IDL for CORBA IIOP, or JMS for Java Messaging Service.

- If ICS is the broker type, the value of the DeliveryTransport property can be MQ, IDL, or JMS, and the default is IDL.
- If the RepositoryDirectory is a local directory, the value may only be JMS.

The connector sends service call requests and administrative messages over CORBA IIOP if the value configured for the DeliveryTransport property is MQ or IDL.

WebSphere MQ and IDL

Use WebSphere MQ rather than IDL for event delivery transport, unless you must have only one product. WebSphere MQ offers the following advantages over IDL:

- Asynchronous communication:
  WebSphere MQ allows the application-specific component to poll and persistently store events even when the server is not available.
- Server side performance:
  WebSphere MQ provides faster performance on the server side. In optimized mode, WebSphere MQ stores only the pointer to an event in the repository database, while the actual event remains in the WebSphere MQ queue. This saves having to write potentially large events to the repository database.
- Agent side performance:
  WebSphere MQ provides faster performance on the application-specific component side. Using WebSphere MQ, the connector's polling thread picks up an event, places it in the connector's queue, then picks up the next event. This is faster than IDL, which requires the connector's polling thread to pick up an event, go over the network into the server process, store the event persistently in the repository database, then pick up the next event.

JMS

Enables communication between the connector and client connector framework using Java Messaging Service (JMS).

If you select JMS as the delivery transport, additional JMS properties such as jms.MessageBrokerName, jms.FactoryClassName, jms.Password, and jms.UserName, appear in Connector Configurator. The first two of these properties are required for this transport.

Important: There may be a memory limitation if you use the JMS transport mechanism for a connector in the following environment:

- AIX 5.0
- WebSphere MQ 5.3.0.1
- When ICS is the integration broker

In this environment, you may experience difficulty starting both the connector controller (on the server side) and the connector (on the client side) due to memory use within the WebSphere MQ client. If your installation uses less than 768M of process heap size, IBM recommends that you set:

- The LDR_CNTRL environment variable in the CWSharedEnv.sh script.

  This script resides in the \bin directory below the product directory. With a text editor, add the following line as the first line in the CWSharedEnv.sh script:

  ```
  export LDR_CNTRL=MAXDATA=0x30000000
  ```

  This line restricts heap memory usage to a maximum of 768 MB (3 segments * 256 MB). If the process memory grows more than this limit, page swapping can occur, which can adversely affect the performance of your system.
- The IPCCBaseAddress property to a value of 11 or 12. For more information on this property, see the *System Installation Guide for UNIX*.

DuplicateEventElimination

When you set this property to true, a JMS-enabled connector can ensure that duplicate events are not delivered to the delivery queue. To use this feature, the connector must have a unique event identifier set as the business object's **ObjectEventId** attribute in the application-specific code. This is done during connector development.

This property can also be set to false.

Note: When DuplicateEventElimination is set to true, you must also configure the MonitorQueue property to enable guaranteed event delivery.

FaultQueue

If the connector experiences an error while processing a message then the connector moves the message to the queue specified in this property, along with a status indicator and a description of the problem.

The default value is CONNECTORNAME/FAULTQUEUE.

JvmMaxHeapSize

The maximum heap size for the agent (in megabytes). This property is applicable only if the RepositoryDirectory value is <REMOTE>.

The default value is 128m.

JvmMaxNativeStackSize

The maximum native stack size for the agent (in kilobytes). This property is applicable only if the RepositoryDirectory value is <REMOTE>.

The default value is 128k.

JvmMinHeapSize

The minimum heap size for the agent (in megabytes). This property is applicable only if the RepositoryDirectory value is <REMOTE>.

The default value is 1m.

jms.FactoryClassName

Specifies the class name to instantiate for a JMS provider. You *must* set this connector property when you choose `JMS` as your delivery transport mechanism (DeliveryTransport).

The default is `CxCommon.Messaging.jms.IBMMQSeriesFactory`.

jms.MessageBrokerName

Specifies the broker name to use for the JMS provider. You *must* set this connector property when you choose `JMS` as your delivery transport mechanism (DeliveryTransport).

The default is `crossworlds.queue.manager`.

jms.NumConcurrentRequests

Specifies the maximum number of concurrent service call requests that can be sent to a connector at the same time. Once that maximum is reached, new service calls block and wait for another request to complete before proceeding.

The default value is 10.

jms.Password

Specifies the password for the JMS provider. A value for this property is optional.

There is no default.

jms.UserName

Specifies the user name for the JMS provider. A value for this property is optional.

There is no default.

ListenerConcurrency

This property supports multi-threading in MQ Listener when ICS is the integration broker. It enables batch writing of multiple events to the database, thus improving system performance. The default value is 1.

This property applies only to connectors using MQ transport. The `DeliveryTransport` property must be set to `MQ`.

Locale

Specifies the language code, country or territory, and, optionally, the associated character code set. The value of this property determines such cultural conventions as collation and sort order of data, date and time formats, and the symbols used in monetary specifications.

A locale name has the following format:

ll_TT.codeset

where:

ll	a two-character language code (usually in lower case)

TT	a two-letter country or territory code (usually in upper case)
codeset	the name of the associated character code set; this portion of the name is often optional.

By default, only a subset of supported locales appears in the drop list. To add other supported values to the drop list, you must manually modify the `\Data\Std\stdConnProps.xml` file in the product directory. For more information, see the appendix on Connector Configurator.

The default value is `en_US`. If the connector has not been globalized, the only valid value for this property is `en_US`. To determine whether a specific connector has been globalized, see the connector version list on these websites:

http://www.ibm.com/software/websphere/wbiadapters/infocenter, or
http://www.ibm.com/websphere/integration/wicserver/infocenter

LogAtInterchangeEnd

Applicable only if RespositoryDirectory is <REMOTE>.

Specifies whether to log errors to the integration broker's log destination. Logging to the broker's log destination also turns on e-mail notification, which generates e-mail messages for the `MESSAGE_RECIPIENT` specified in the `InterchangeSystem.cfg` file when errors or fatal errors occur.

For example, when a connector loses its connection to its application, if `LogAtInterChangeEnd` is set to `true`, an e-mail message is sent to the specified message recipient. The default is `false`.

MaxEventCapacity

The maximum number of events in the controller buffer. This property is used by flow control and is applicable only if the value of the `RepositoryDirectory` property is <REMOTE>.

The value can be a positive integer between 1 and 2147483647. The default value is 2147483647.

MessageFileName

The name of the connector message file. The standard location for the message file is `\connectors\messages`. Specify the message filename in an absolute path if the message file is not located in the standard location.

If a connector message file does not exist, the connector uses `InterchangeSystem.txt` as the message file. This file is located in the product directory.

Note: To determine whether a specific connector has its own message file, see the individual adapter user guide.

MonitorQueue

The logical queue that the connector uses to monitor duplicate events. It is used only if the `DeliveryTransport` property value is JMS and `DuplicateEventElimination` is set to `TRUE`.

The default value is CONNECTORNAME/MONITORQUEUE

OADAutoRestartAgent

Valid only when the RepositoryDirectory is <REMOTE>.

Specifies whether the connector uses the automatic and remote restart feature. This feature uses the MQ-triggered Object Activation Daemon (OAD) to restart the connector after an abnormal shutdown, or to start a remote connector from System Monitor.

This property must be set to trueto enable the automatic and remote restart feature. For information on how to configure the MQ-triggered OAD feature. see the *Installation Guide for Windows* or *for UNIX*.

The default value is false.

OADMaxNumRetry

Valid only when the RepositoryDirectory is <REMOTE>.

Specifies the maximum number of times that the MQ-triggered OAD automatically attempts to restart the connector after an abnormal shutdown. The OADAutoRestartAgent property must be set to true for this property to take effect.

The default value is 1000.

OADRetryTimeInterval

Valid only when the RepositoryDirectory is <REMOTE>.

Specifies the number of minutes in the retry-time interval for the MQ-triggered OAD. If the connector agent does not restart within this retry-time interval, the connector controller asks the OAD to restart the connector agent again. The OAD repeats this retry process as many times as specified by the OADMaxNumRetry property. The OADAutoRestartAgent property must be set to true for this property to take effect.

The default is 10.

PollEndTime

Time to stop polling the event queue. The format is HH:MM, where *HH* represents 0-23 hours, and *MM* represents 0-59 seconds.

You must provide a valid value for this property. The default value is HH:MM, but must be changed.

PollFrequency

The amount of time between polling actions. Set PollFrequency to one of the following values:

- The number of milliseconds between polling actions.
- The word key, which causes the connector to poll only when you type the letter p in the connector's Command Prompt window. Enter the word in lowercase.
- The word no, which causes the connector not to poll. Enter the word in lowercase.

The default is 10000.

Important: Some connectors have restrictions on the use of this property. To determine whether a specific connector does, see the installing and configuring chapter of its adapter guide.

PollQuantity

Designates the number of items from the application that the connector should poll for. If the adapter has a connector-specific property for setting the poll quantity, the value set in the connector-specific property will override the standard property value.

PollStartTime

The time to start polling the event queue. The format is *HH:MM*, where *HH* represents 0-23 hours, and *MM* represents 0-59 seconds.

You must provide a valid value for this property. The default value is HH:MM, but must be changed.

RequestQueue

The queue that is used by the integration broker to send business objects to the connector.

The default value is CONNECTOR/REQUESTQUEUE.

RepositoryDirectory

The location of the repository from which the connector reads the XML schema documents that store the meta-data for business object definitions.

When the integration broker is ICS, this value must be set to <REMOTE> because the connector obtains this information from the InterChange Server repository.

When the integration broker is a WebSphere message broker or WAS, this value must be set to <*local directory*>.

ResponseQueue

Applicable only if DeliveryTransport is JMS and required only if RepositoryDirectory is <REMOTE>.

Designates the JMS response queue, which delivers a response message from the connector framework to the integration broker. When the integration broker is ICS, the server sends the request and waits for a response message in the JMS response queue.

RestartRetryCount

Specifies the number of times the connector attempts to restart itself. When used for a parallel connector, specifies the number of times the master connector application-specific component attempts to restart the slave connector application-specific component.

The default is 3.

RestartRetryInterval

Specifies the interval in minutes at which the connector attempts to restart itself. When used for a parallel connector, specifies the interval at which the master connector application-specific component attempts to restart the slave connector application-specific component. Possible values ranges from 1 to 2147483647.

The default is 1.

RHF2MessageDomain

WebSphere message brokers and WAS only.

This property allows you to configure the value of the field domain name in the JMS header. When data is sent to WMQI over JMS transport, the adapter framework writes JMS header information, with a domain name and a fixed value of `mrm`. A connfigurable domain name enables users to track how the WMQI broker processes the message data.

A sample header would look like this:

```
<mcd><Msd>mrm</Msd><Set>3</Set><Type>
Retek_POPhyDesc</Type><Fmt>CwXML</Fmt></mcd>
```

The default value is `mrm`, but it may also be set to `xml`. This property only appears when DeliveryTransport is set to `JMS`and WireFormat is set to `CwXML`.

SourceQueue

Applicable only if `DeliveryTransport` is JMS and ContainerManagedEvents is specified.

Designates the JMS source queue for the connector framework in support of guaranteed event delivery for JMS-enabled connectors that use a JMS event store. For further information, see "ContainerManagedEvents" on page 58.

The default value is `CONNECTOR/SOURCEQUEUE`.

SynchronousRequestQueue

Applicable only if `DeliveryTransport` is JMS.

Delivers request messages that require a synchronous response from the connector framework to the broker. This queue is necessary only if the connector uses synchronous execution. With synchronous execution, the connector framework sends a message to the `SynchronousRequestQueue` and waits for a response back from the broker on the `SynchronousResponseQueue`. The response message sent to the connector bears a correlation ID that matches the ID of the original message.

The default is `CONNECTORNAME/SYNCHRONOUSREQUESTQUEUE`

SynchronousResponseQueue

Applicable only if `DeliveryTransport` is JMS.

Delivers response messages sent in reply to a synchronous request from the broker to the connector framework. This queue is necessary only if the connector uses synchronous execution.

The default is CONNECTORNAME/SYNCHRONOUSRESPONSEQUEUE

SynchronousRequestTimeout

Applicable only if DeliveryTransport is JMS.

Specifies the time in minutes that the connector waits for a response to a synchronous request. If the response is not received within the specified time, then the connector moves the original synchronous request message into the fault queue along with an error message.

The default value is 0.

WireFormat

Message format on the transport.

- If the RepositoryDirectory is a local directory, the setting is CwXML.
- If the value of RepositoryDirectory is <REMOTE>, the setting isCwBO.

WsifSynchronousRequest Timeout

WAS integration broker only.

Specifies the time in minutes that the connector waits for a response to a synchronous request. If the response is not received within the specified, time then the connector moves the original synchronous request message into the fault queue along with an error message.

The default value is 0.

XMLNameSpaceFormat

WebSphere message brokers and WAS integration broker only.

A strong property that allows the user to specify short and long name spaces in the XML format of business object definitions.

The default value is short.

Appendix B. Connector Configurator

This appendix describes how to use Connector Configurator to set configuration property values for your adapter.

You use Connector Configurator to:

- Create a connector-specific property template for configuring your connector
- Create a configuration file
- Set properties in a configuration file

Note:

In this document, backslashes (\) are used as the convention for directory paths. For UNIX installations, substitute slashes (/) for backslashes and follow the conventions for each operating system.

The topics covered in this appendix are:

Overview of Connector Configurator

Connector Configurator allows you to configure the connector component of your adapter for use with these integration brokers:

- WebSphere InterChange Server (ICS)
- WebSphere MQ Integrator, WebSphere MQ Integrator Broker, and WebSphere Business Integration Message Broker, collectively referred to as the WebSphere Message Brokers (WMQI)
- WebSphere Application Server (WAS)

You use Connector Configurator to:

- Create a **connector-specific property template** for configuring your connector.
- Create a **connector configuration file**; you must create one configuration file for each connector you install.
- Set properties in a configuration file.
  You may need to modify the default values that are set for properties in the connector templates. You must also designate supported business object definitions and, with ICS, maps for use with collaborations as well as specify messaging, logging and tracing, and data handler parameters, as required.

The mode in which you run Connector Configurator, and the configuration file type you use, may differ according to which integration broker you are running. For example, if WMQI is your broker, you run Connector Configurator directly, and not from within System Manager (see "Running Configurator in stand-alone mode" on page 68).

© Copyright IBM Corp. 2003, 2004

Connector configuration properties include both standard configuration properties (the properties that all connectors have) and connector-specific properties (properties that are needed by the connector for a specific application or technology).

Because **standard properties** are used by all connectors, you do not need to define those properties from scratch; Connector Configurator incorporates them into your configuration file as soon as you create the file. However, you do need to set the value of each standard property in Connector Configurator.

The range of standard properties may not be the same for all brokers and all configurations. Some properties are available only if other properties are given a specific value. The Standard Properties window in Connector Configurator will show the properties available for your particular configuration.

For **connector-specific properties**, however, you need first to define the properties and then set their values. You do this by creating a connector-specific property template for your particular adapter. There may already be a template set up in your system, in which case, you simply use that. If not, follow the steps in "Creating a new template" on page 69 to set up a new one.

Note: Connector Configurator runs only in a Windows environment. If you are running the connector in a UNIX environment, use Connector Configurator in Windows to modify the configuration file and then copy the file to your UNIX environment.

Starting Connector Configurator

You can start and run Connector Configurator in either of two modes:

- Independently, in stand-alone mode
- From System Manager

Running Configurator in stand-alone mode

You can run Connector Configurator independently and work with connector configuration files, irrespective of your broker.

To do so:

- From **Start>Programs**, click **IBM WebSphere InterChange Server>IBM WebSphere Business Integration Toolset>Development>Connector Configurator**.
- Select **File>New>Configuration File**.
- When you click the pull-down menu next to **System Connectivity Integration Broker**, you can select ICS, WebSphere Message Brokers or WAS, depending on your broker.

You may choose to run Connector Configurator independently to generate the file, and then connect to System Manager to save it in a System Manager project (see "Completing a configuration file" on page 73.)

Running Configurator from System Manager

You can run Connector Configurator from System Manager.

To run Connector Configurator:

1. Open the System Manager.
2. In the System Manager window, expand the **Integration Component Libraries** icon and highlight **Connectors**.
3. From the System Manager menu bar, click **Tools>Connector Configurator**. The Connector Configurator window opens and displays a **New Connector** dialog box.
4. When you click the pull-down menu next to **System Connectivity Integration Broker**, you can select ICS, WebSphere Message Brokers or WAS, depending on your broker.

To edit an existing configuration file:

1. In the System Manager window, select any of the configuration files listed in the Connector folder and right-click on it. Connector Configurator opens and displays the configuration file with the integration broker type and file name at the top.
2. Click the Standard Properties tab to see which properties are included in this configuration file.

Creating a connector-specific property template

To create a configuration file for your connector, you need a connector-specific property template as well as the system-supplied standard properties.

You can create a brand-new template for the connector-specific properties of your connector, or you can use an existing file as the template.

- To create a new template, see "Creating a new template" on page 69.
- To use an existing file, simply modify an existing template and save it under the new name.

Creating a new template

This section describes how you create properties in the template, define general characteristics and values for those properties, and specify any dependencies between the properties. Then you save the template and use it as the base for creating a new connector configuration file.

To create a template:

1. Click **File>New>Connector-Specific Property Template**.
2. The **Connector-Specific Property Template** dialog box appears, with the following fields:
   - **Template**, and **Name**

     Enter a unique name that identifies the connector, or type of connector, for which this template will be used. You will see this name again when you open the dialog box for creating a new configuration file from a template.
   - **Old Template**, and **Select the Existing Template to Modify**

     The names of all currently available templates are displayed in the Template Name display.

- To see the connector-specific property definitions in any template, select that template's name in the **Template Name** display. A list of the property definitions contained in that template will appear in the **Template Preview** display. You can use an existing template whose property definitions are similar to those required by your connector as a starting point for your template.

3. Select a template from the **Template Name** display, enter that template name in the **Find Name** field (or highlight your selection in **Template Name**), and click **Next**.

If you do not see any template that displays the connector-specific properties used by your connector, you will need to create one.

Specifying general characteristics

When you click **Next** to select a template, the **Properties - Connector-Specific Property Template** dialog box appears. The dialog box has tabs for General characteristics of the defined properties and for Value restrictions. The General display has the following fields:

- **General:**
  Property Type
  Updated Method
  Description
- **Flags**
  Standard flags
- **Custom Flag**
  Flag

After you have made selections for the general characteristics of the property, click the **Value** tab.

Specifying values

The **Value** tab enables you to set the maximum length, the maximum multiple values, a default value, or a value range for the property. It also allows editable values. To do so:

1. Click the **Value** tab. The display panel for Value replaces the display panel for General.
2. Select the name of the property in the **Edit properties** display.
3. In the fields for **Max Length** and **Max Multiple Values**, make any changes. The changes will not be accepted unless you also open the **Property Value** dialog box for the property, described in the next step.
4. Right-click the box in the top left-hand corner of the value table and click **Add**. A **Property Value** dialog box appears. Depending on the property type, the dialog box allows you to enter either a value, or both a value and range. Enter the appropriate value or range, and click **OK**.
5. The **Value** panel refreshes to display any changes you made in **Max Length** and **Max Multiple Values**. It displays a table with three columns:

   The **Value** column shows the value that you entered in the **Property Value** dialog box, and any previous values that you created.

   The **Default Value** column allows you to designate any of the values as the default.

   The **Value Range** shows the range that you entered in the **Property Value** dialog box.

After a value has been created and appears in the grid, it can be edited from within the table display. To make a change in an existing value in the table, select an entire row by clicking on the row number. Then right-click in the **Value** field and click **Edit Value**.

Setting dependencies

When you have made your changes to the **General** and **Value** tabs, click **Next**. The **Dependences - Connector-Specific Property Template** dialog box appears.

A dependent property is a property that is included in the template and used in the configuration file *only if* the value of another property meets a specific condition. For example, `PollQuantity` appears in the template only if JMS is the transport mechanism and `DuplicateEventElimination` is set to `True`.
To designate a property as dependent and to set the condition upon which it depends, do this:

1. In the **Available Properties** display, select the property that will be made dependent.
2. In the **Select Property** field, use the drop-down menu to select the property that will hold the conditional value.
3. In the **Condition Operator** field, select one of the following:

   == (equal to)

   != (not equal to)

   > (greater than)

   < (less than)

   >= (greater than or equal to)

   <=(less than or equal to)
4. In the **Conditional Value** field, enter the value that is required in order for the dependent property to be included in the template.
5. With the dependent property highlighted in the **Available Properties** display, click an arrow to move it to the **Dependent Property** display.
6. Click **Finish**. Connector Configurator stores the information you have entered as an XML document, under `\data\app` in the`\bin` directory where you have installed Connector Configurator.

Creating a new configuration file

When you create a new configuration file, your first step is to select an integration broker. The broker you select determines the properties that will appear in the configuration file.

To select a broker:

- In the Connector Configurator home menu, click **File>New>Connector Configuration**. The **New Connector** dialog box appears.
- In the I**ntegration Broker** field, select ICS, WebSphere Message Brokers or WAS connectivity.
- Complete the remaining fields in the **New Connector** window, as described later in this chapter.

You can also do this:

- In the System Manager window, right-click on the **Connectors** folder and select **Create New Connector**. Connector Configurator opens and displays the **New Connector** dialog box.

Creating a configuration file from a connector-specific template

Once a connector-specific template has been created, you can use it to create a configuration file:

1. Click **File>New>Connector Configuration**.
2. The **New Connector** dialog box appears, with the following fields:
   - **Name**

     Enter the name of the connector. Names are case-sensitive. The name you enter must be unique, and must be consistent with the file name for a connector that is installed on the system.

     Important: Connector Configurator does not check the spelling of the name that you enter. You must ensure that the name is correct.
   - **System Connectivity**

     Click ICS or WebSphere Message Brokers or WAS.
   - **Select Connector-Specific Property Template**

     Type the name of the template that has been designed for your connector. The available templates are shown in the **Template Name** display. When you select a name in the Template Name display, the **Property Template Preview** display shows the connector-specific properties that have been defined in that template.

     Select the template you want to use and click **OK**.
3. A configuration screen appears for the connector that you are configuring. The title bar shows the integration broker and connector names. You can fill in all the field values to complete the definition now, or you can save the file and complete the fields later.
4. To save the file, click **File>Save>To File** or **File>Save>To Project**. To save to a project, System Manager must be running.
   If you save as a file, the **Save File Connector** dialog box appears. Choose `*.cfg` as the file type, verify in the File Name field that the name is spelled correctly and has the correct case, navigate to the directory where you want to locate the file, and click **Save**. The status display in the message panel of Connector Configurator indicates that the configuration file was successfully created.

   Important: The directory path and name that you establish here must match the connector configuration file path and name that you supply in the startup file for the connector.
5. To complete the connector definition, enter values in the fields for each of the tabs of the Connector Configurator window, as described later in this chapter.

Using an existing file

You may have an existing file available in one or more of the following formats:

- A connector definition file.
  This is a text file that lists properties and applicable default values for a specific connector. Some connectors include such a file in a `\repository` directory in their delivery package (the file typically has the extension `.txt`; for example, CN_XML.txt for the XML connector).
- An ICS repository file.
  Definitions used in a previous ICS implementation of the connector may be available to you in a repository file that was used in the configuration of that connector. Such a file typically has the extension `.in` or `.out`.

- A previous configuration file for the connector.
  Such a file typically has the extension *.cfg.

Although any of these file sources may contain most or all of the connector-specific properties for your connector, the connector configuration file will not be complete until you have opened the file and set properties, as described later in this chapter.

To use an existing file to configure a connector, you must open the file in Connector Configurator, revise the configuration, and then resave the file.

Follow these steps to open a *.txt, *.cfg, or *.in file from a directory:

1. In Connector Configurator, click **File>Open>From File**.
2. In the **Open File Connector** dialog box, select one of the following file types to see the available files:
   - Configuration (`*.cfg`)
   - ICS Repository (`*.in`, `*.out`)

     Choose this option if a repository file was used to configure the connector in an ICS environment. A repository file may include multiple connector definitions, all of which will appear when you open the file.
   - All files (*.*)

     Choose this option if a `*.txt` file was delivered in the adapter package for the connector, or if a definition file is available under another extension.
3. In the directory display, navigate to the appropriate connector definition file, select it, and click **Open**.

Follow these steps to open a connector configuration from a System Manager project:

1. Start System Manager. A configuration can be opened from or saved to System Manager only if System Manager has been started.
2. Start Connector Configurator.
3. Click **File>Open>From Project**.

Completing a configuration file

When you open a configuration file or a connector from a project, the Connector Configurator window displays the configuration screen, with the current attributes and values.

The title of the configuration screen displays the integration broker and connector name as specified in the file. Make sure you have the correct broker. If not, change the broker value before you configure the connector. To do so:

1. Under the **Standard Properties** tab, select the value field for the `BrokerType` property. In the drop-down menu, select the value `ICS`, `WMQI`, or `WAS`.
2. The Standard Properties tab will display the properties associated with the selected broker. You can save the file now or complete the remaining configuration fields, as described in “Specifying supported business object definitions” on page 76..
3. When you have finished your configuration, click **File>Save>To Project** or **File>Save>To File**.

   If you are saving to file, select `*.cfg` as the extension, select the correct location for the file and click **Save**.

If multiple connector configurations are open, click **Save All to File** to save all of the configurations to file, or click **Save All to Project** to save all connector configurations to a System Manager project.

Before it saves the file, Connector Configurator checks that values have been set for all required standard properties. If a required standard property is missing a value, Connector Configurator displays a message that the validation failed. You must supply a value for the property in order to save the configuration file.

Setting the configuration file properties

When you create and name a new connector configuration file, or when you open an existing connector configuration file, Connector Configurator displays a configuration screen with tabs for the categories of required configuration values.

Connector Configurator requires values for properties in these categories for connectors running on all brokers:

- Standard Properties
- Connector-specific Properties
- Supported Business Objects
- Trace/Log File values
- Data Handler (applicable for connectors that use JMS messaging with guaranteed event delivery)

Note: For connectors that use JMS messaging, an additional category may display, for configuration of data handlers that convert the data to business objects.

For connectors running on **ICS**, values for these properties are also required:

- Associated Maps
- Resources
- Messaging (where applicable)

Important: Connector Configurator accepts property values in either English or non-English character sets. However, the names of both standard and connector-specific properties, and the names of supported business objects, must use the English character set only.

Standard properties differ from connector-specific properties as follows:

- Standard properties of a connector are shared by both the application-specific component of a connector and its broker component. All connectors have the same set of standard properties. These properties are described in Appendix A of each adapter guide. You can change some but not all of these values.
- Application-specific properties apply only to the application-specific component of a connector, that is, the component that interacts directly with the application. Each connector has application-specific properties that are unique to its application. Some of these properties provide default values and some do not; you can modify some of the default values. The installation and configuration chapters of each adapter guide describe the application-specific properties and the recommended values.

The fields for **Standard Properties** and **Connector-Specific Properties** are color-coded to show which are configurable:

- A field with a grey background indicates a standard property. You can change the value but cannot change the name or remove the property.
- A field with a white background indicates an application-specific property. These properties vary according to the specific needs of the application or connector. You can change the value and delete these properties.
- Value fields are configurable.
- The **Update Method** field is informational and not configurable. This field specifies the action required to activate a property whose value has changed.

Setting standard connector properties

To change the value of a standard property:

1. Click in the field whose value you want to set.
2. Either enter a value, or select one from the drop-down menu if it appears.
3. After entering all the values for the standard properties, you can do one of the following:
   - To discard the changes, preserve the original values, and exit Connector Configurator, click **File>Exit** (or close the window), and click **No** when prompted to save changes.
   - To enter values for other categories in Connector Configurator, select the tab for the category. The values you enter for **Standard Properties** (or any other category) are retained when you move to the next category. When you close the window, you are prompted to either save or discard the values that you entered in all the categories as a whole.
   - To save the revised values, click **File>Exit** (or close the window) and click **Yes** when prompted to save changes. Alternatively, click **Save>To File** from either the File menu or the toolbar.

Setting application-specific configuration properties

For application-specific configuration properties, you can add or change property names, configure values, delete a property, and encrypt a property. The default property length is 255 characters.

1. Right-click in the top left portion of the grid. A pop-up menu bar will appear. Click **Add** to add a property. To add a child property, right-click on the parent row number and click **Add child**.
2. Enter a value for the property or child property.
3. To encrypt a property, select the **Encrypt** box.
4. Choose to save or discard changes, as described for "Setting standard connector properties."

The Update Method displayed for each property indicates whether a component or agent restart is necessary to activate changed values.

Important: Changing a preset application-specific connector property name may cause a connector to fail. Certain property names may be needed by the connector to connect to an application or to run properly.

Encryption for connector properties

Application-specific properties can be encrypted by selecting the **Encrypt** check box in the **Edit Property** window. To decrypt a value, click to clear the **Encrypt** check box, enter the correct value in the **Verification** dialog box, and click **OK**. If the entered value is correct, the value is decrypted and displays.

The adapter user guide for each connector contains a list and description of each property and its default value.

If a property has multiple values, the **Encrypt** check box will appear for the first value of the property. When you select **Encrypt**, all values of the property will be encrypted. To decrypt multiple values of a property, click to clear the **Encrypt** check box for the first value of the property, and then enter the new value in the **Verification** dialog box. If the input value is a match, all multiple values will decrypt.

Update method

Refer to the descriptions of update methods found in the *Standard configuration properties for connectors* appendix, under "Setting and updating property values" on page 52.

Specifying supported business object definitions

Use the **Supported Business Objects** tab in Connector Configurator to specify the business objects that the connector will use. You must specify both generic business objects and application-specific business objects, and you must specify associations for the maps between the business objects.

Note: Some connectors require that certain business objects be specified as supported in order to perform event notification or additional configuration (using meta-objects) with their applications. For more information, see the *Connector Development Guide for C++* or the *Connector Development Guide for Java*.

If ICS is your broker

To specify that a business object definition is supported by the connector, or to change the support settings for an existing business object definition, click the **Supported Business Objects** tab and use the following fields.

Business object name: To designate that a business object definition is supported by the connector, with System Manager running:

1. Click an empty field in the **Business Object Name** list. A drop-down list displays, showing all the business object definitions that exist in the System Manager project.
2. Click on a business object to add it.
3. Set the **Agent Support** (described below) for the business object.
4. In the File menu of the Connector Configurator window, click **Save to Project**. The revised connector definition, including designated support for the added business object definition, is saved to the project in System Manager.

To delete a business object from the supported list:

1. To select a business object field, click the number to the left of the business object.
2. From the **Edit** menu of the Connector Configurator window, click **Delete Row**. The business object is removed from the list display.
3. From the **File** menu, click **Save to Project**.

Deleting a business object from the supported list changes the connector definition and makes the deleted business object unavailable for use in this implementation of this connector. It does not affect the connector code, nor does it remove the business object definition itself from System Manager.

Agent support: If a business object has Agent Support, the system will attempt to use that business object for delivering data to an application via the connector agent.

Typically, application-specific business objects for a connector are supported by that connector's agent, but generic business objects are not.

To indicate that the business object is supported by the connector agent, check the **Agent Support** box. The Connector Configurator window does not validate your Agent Support selections.

Maximum transaction level: The maximum transaction level for a connector is the highest transaction level that the connector supports.

For most connectors, `Best Effort` is the only possible choice.

You must restart the server for changes in transaction level to take effect.

If a WebSphere Message Broker is your broker

If you are working in stand-alone mode (not connected to System Manager), you must enter the business name manually.

If you have System Manager running, you can select the empty box under the **Business Object Name** column in the **Supported Business Objects** tab. A combo box appears with a list of the business object available from the Integration Component Library project to which the connector belongs. Select the business object you want from the list.

The **Message Set ID** is an optional field for WebSphere Business Integration Message Broker 5.0, and need not be unique if supplied. However, for WebSphere MQ Integrator and Integrator Broker 2.1, you must supply a unique **ID**.

If WAS is your broker

When WebSphere Application Server is selected as your broker type, Connector Configurator does not require message set IDs. The **Supported Business Objects** tab shows a **Business Object Name** column only for supported business objects.

If you are working in stand-alone mode (not connected to System Manager), you must enter the business object name manually.

If you have System Manager running, you can select the empty box under the Business Object Name column in the Supported Business Objects tab. A combo box appears with a list of the business objects available from the Integration Component Library project to which the connector belongs. Select the business object you want from this list.

Associated maps (ICS only)

Each connector supports a list of business object definitions and their associated maps that are currently active in WebSphere InterChange Server. This list appears when you select the **Associated Maps** tab.

The list of business objects contains the application-specific business object which the agent supports and the corresponding generic object that the controller sends to the subscribing collaboration. The association of a map determines which map

will be used to transform the application-specific business object to the generic business object or the generic business object to the application-specific business object.

If you are using maps that are uniquely defined for specific source and destination business objects, the maps will already be associated with their appropriate business objects when you open the display, and you will not need (or be able) to change them.

If more than one map is available for use by a supported business object, you will need to explicitly bind the business object with the map that it should use.

The **Associated Maps** tab displays the following fields:

- **Business Object Name**

  These are the business objects supported by this connector, as designated in the **Supported Business Objects** tab. If you designate additional business objects under the Supported Business Objects tab, they will be reflected in this list after you save the changes by choosing **Save to Project** from the **File** menu of the Connector Configurator window.
- **Associated Maps**

  The display shows all the maps that have been installed to the system for use with the supported business objects of the connector. The source business object for each map is shown to the left of the map name, in the **Business Object Name** display.
- **Explicit**

  In some cases, you may need to explicitly bind an associated map.

  Explicit binding is required only when more than one map exists for a particular supported business object. When ICS boots, it tries to automatically bind a map to each supported business object for each connector. If more than one map takes as its input the same business object, the server attempts to locate and bind one map that is the superset of the others.

  If there is no map that is the superset of the others, the server will not be able to bind the business object to a single map, and you will need to set the binding explicitly.

  To explicitly bind a map:

  1. In the **Explicit** column, place a check in the check box for the map you want to bind.
  2. Select the map that you intend to associate with the business object.
  3. In the **File** menu of the Connector Configurator window, click **Save to Project**.
  4. Deploy the project to ICS.
  5. Reboot the server for the changes to take effect.

Resources (ICS)

The **Resource** tab allows you to set a value that determines whether and to what extent the connector agent will handle multiple processes concurrently, using connector agent parallelism.
Not all connectors support this feature. If you are running a connector agent that was designed in Java to be multi-threaded, you are advised not to use this feature, since it is usually more efficient to use multiple threads than multiple processes.

Messaging (ICS)

The messaging properties are available only if you have set MQ as the value of the `DeliveryTransport` standard property and ICS as the broker type. These properties affect how your connector will use queues.

Setting trace/log file values

When you open a connector configuration file or a connector definition file, Connector Configurator uses the logging and tracing values of that file as default values. You can change those values in Connector Configurator.

To change the logging and tracing values:

1. Click the **Trace/Log Files** tab.
2. For either logging or tracing, you can choose to write messages to one or both of the following:
   - To console (STDOUT):
     Writes logging or tracing messages to the STDOUT display.

     Note: You can only use the STDOUT option from the **Trace/Log Files** tab for connectors running on the Windows platform.
   - To File:
     Writes logging or tracing messages to a file that you specify. To specify the file, click the directory button (ellipsis), navigate to the preferred location, provide a file name, and click **Save**. Logging or tracing message are written to the file and location that you specify.

     Note: Both logging and tracing files are simple text files. You can use the file extension that you prefer when you set their file names. For tracing files, however, it is advisable to use the extension `.trace` rather than `.trc`, to avoid confusion with other files that might reside on the system. For logging files, `.log` and `.txt` are typical file extensions.

Data handlers

The data handlers section is available for configuration only if you have designated a value of JMS for DeliveryTransport and a value of JMS for ContainerManagedEvents. Not all adapters make use of data handlers.

See the descriptions under `ContainerManagedEvents` in Appendix A, Standard Properties, for values to use for these properties. For additional details, see the *Connector Development Guide for C++* or the *Connector Development Guide for Java.*

Saving your configuration file

When you have finished configuring your connector, save the connector configuration file. Connector Configurator saves the file in the broker mode that you selected during configuration. The title bar of Connector Configurator always displays the broker mode (ICS, WMQI or WAS) that it is currently using.

The file is saved as an XML document. You can save the XML document in three ways:

- From System Manager, as a file with a `*.con` extension in an Integration Component Library, or
- In a directory that you specify.
- In stand-alone mode, as a file with a `*.cfg` extension in a directory folder.

For details about using projects in System Manager, and for further information about deployment, see the following implementation guides:

- For ICS: *Implementation Guide for WebSphere InterChange Server*
- For WebSphere Message Brokers: *Implementing Adapters with WebSphere Message Brokers*
- For WAS: *Implementing Adapters with WebSphere Application Server*

Changing a configuration file

You can change the integration broker setting for an existing configuration file. This enables you to use the file as a template for creating a new configuration file, which can be used with a different broker.

Note: You will need to change other configuration properties as well as the broker mode property if you switch integration brokers.

To change your broker selection within an existing configuration file (optional):

- Open the existing configuration file in Connector Configurator.
- Select the **Standard Properties** tab.
- In the **BrokerType** field of the Standard Properties tab, select the value that is appropriate for your broker.
  When you change the current value, the available tabs and field selections on the properties screen will immediately change, to show only those tabs and fields that pertain to the new broker you have selected.

Completing the configuration

After you have created a configuration file for a connector and modified it, make sure that the connector can locate the configuration file when the connector starts up.

To do so, open the startup file used for the connector, and verify that the location and file name used for the connector configuration file match exactly the name you have given the file and the directory or path where you have placed it.

Using Connector Configurator in a globalized environment

Connector Configurator is globalized and can handle character conversion between the configuration file and the integration broker. Connector Configurator uses native encoding. When it writes to the configuration file, it uses UTF-8 encoding.

Connector Configurator supports non-English characters in:

- All value fields
- Log file and trace file path (specified in the **Trace/Log files** tab)

The drop list for the `CharacterEncoding` and `Locale` standard configuration properties displays only a subset of supported values. To add other values to the drop list, you must manually modify the `\Data\Std\stdConnProps.xml` file in the product directory.

For example, to add the locale `en_GB` to the list of values for the `Locale` property, open the `stdConnProps.xml` file and add the line in boldface type below:

```
<Property name="Locale"
isRequired="true"
updateMethod="component restart">
                <ValidType>String</ValidType>
            <ValidValues>
                                <Value>ja_JP</Value>
                                <Value>ko_KR</Value>
                                <Value>zh_CN</Value>
                                <Value>zh_TW</Value>
                                <Value>fr_FR</Value>
                                <Value>de_DE</Value>
                                <Value>it_IT</Value>
                                <Value>es_ES</Value>
                                <Value>pt_BR</Value>
                                <Value>en_US</Value>
                                <Value>en_GB</Value>
                    <DefaultValue>en_US</DefaultValue>
            </ValidValues>
    </Property>
```

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

© Copyright IBM Corp. 2003, 2004

Burlingame, CA 94010
U.S.A

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

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

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

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

This information may contain examples of data and reports used in daily business operations. To illustrate them as completely as possible, the examples may include the names of individuals, companies, brands, and products. All of these names are fictitious and any similarity to the names and addresses used by an actual business enterprise is entirely coincidental.

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

Programming interface information

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

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

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

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

Trademarks and service marks

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

IBM
the IBM logo
AIX
CrossWorlds
DB2
DB2 Universal Database
Domino
Lotus
Lotus Notes
MQIntegrator
MQSeries
Tivoli
WebSphere

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

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

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

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

The adapter for EJB includes software developed by the Eclipse Project (http://www.eclipse.org/).

WebSphere Business Integration Adapter Framework V2.4.0

Printed in USA