

IBM WebSphere Business Integration Adapters



Adapter for CORBA User Guide

Version 1.2.x

IBM WebSphere Business Integration Adapters



Adapter for CORBA User Guide

Version 1.2.x

Note!

Before using this information and the product it supports, read the information in "Notices" on page 103.

25June2004

This edition of this document applies to IBM WebSphere Business Integration adapter for CORBA, version 1.2.x.

To send us your comments about IBM WebSphere Business Integration documentation, 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

About this document	v
Audience	v
Prerequisites for this document	v
Related documents	v
Typographic conventions	vi
New in this release	vii
New in Release 1.2.x	vii
New in Release 1.1.x	vii
Chapter 1. Overview	1
Adapter environment	1
Terminology	5
Architecture of the connector running as a client	6
Architecture of the connector running as a server	11
Business object requests	14
Verb processing	14
Custom business object handlers	15
Chapter 2. Installing the adapter	17
Overview of installation tasks	17
Connector file structure	18
Post-installation tasks	19
Chapter 3. Configuring the adapter	21
Overview of configuration tasks	21
Configuring the connector	21
Creating multiple connector instances	31
Starting the connector	32
Stopping the connector	33
Using log and trace files	33
Chapter 4. Understanding business objects	35
Defining metadata	35
Connector business object structure	36
Mapping attributes: CORBA, Java, and business object	44
Sample business object properties	45
Generating business objects	49
Chapter 5. Creating and modifying business objects	51
Overview of the ODA for CORBA	51
IDL file compatibility	51
Generating business object definitions	52
Specifying business object information	57
Uploading business object files	62
Chapter 6. Troubleshooting and error handling	63
Error handling	63
Troubleshooting tips	66
Logging	66
Tracing	66
Appendix A. Standard configuration properties for connectors	69

New and deleted properties	69
Configuring standard connector properties	69
Summary of standard properties	70
Standard configuration properties	75
Appendix B. Connector Configurator	87
Overview of Connector Configurator	87
Starting Connector Configurator	88
Running Configurator from System Manager	88
Creating a connector-specific property template	89
Creating a new configuration file	91
Using an existing file	92
Completing a configuration file.	93
Setting the configuration file properties	94
Saving your configuration file	99
Changing a configuration file	100
Completing the configuration	100
Using Connector Configurator in a globalized environment	100
Notices	103
Programming interface information	104
Trademarks and service marks	105

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 the installation, configuration, business object development, and troubleshooting for the IBM WebSphere Business Integration adapter for CORBA.

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 CORBA technology.

Related documents

The complete set of documentation available with this product describes the features and components common to all WebSphere Business Integration Adapters 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.

Note: Important information about this product may be available in Technical Support Technotes and Flashes issued after this document was published. These can be found on the WebSphere Business Integration Support Web

site, <http://www.ibm.com/software/integration/websphere/support/>. Select the component area of interest and browse the Technotes and Flashes sections. Additional information might also be available in IBM Redbooks at <http://www.redbooks.ibm.com/>.

Typographic conventions

This document uses the following conventions:

<code>courier font</code>	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.
<i>italic, italic</i>	Indicates a variable name or a cross-reference.
<i>blue outline</i>	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, <code>option[,...]</code> 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 <code><server_name><connector_name>tmp.log</code> .
/, \	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 product is installed on your system.
<code>%text%</code> and <code>\$text</code>	Text within percent (%) signs indicates the value of the Windows™ text system variable or user variable. The equivalent notation in a UNIX environment is <code>\$text</code> , indicating the value of the text UNIX environment variable.
<i>ProductDir</i>	Represents the directory where the product is installed.

New in this release

New in Release 1.2.x

Updated in June 2004. For version 1.2.x of the adapter for CORBA, the following items are new in this release:

- As of version 1.2.x, the adapter for CORBA is not supported on Solaris 7, so references to that platform version have been deleted from this guide.
- Beginning with the 1.2.x version, during installation of the adapter for CORBA, Windows and AIX users are required to specify the directory pathname where the IBM JDK 1.3.1 SR5 is located (as explained in “JDK software” on page 3, this software is a prerequisite for installing the connector for CORBA). For details, see “Windows and AIX installation” on page 17.
- Sample configuration scenarios for the connector running as a client and as a server are provided in Chapter 3, “Configuring the adapter,” on page 21. For details, see “Sample configuration scenarios” on page 28.

New in Release 1.1.x

Updated in December 2003. For version 1.1.x of the adapter for CORBA, the following items are new in this release:

- Adapter installation information has been moved from this guide. See “Install the adapter for CORBA and related files” on page 17 for the new location of that information.
- Beginning with the 1.1.x version, the adapter for CORBA is no longer supported on Microsoft Windows NT.
- Beginning with the 1.1.x version, the adapter for CORBA supports IBM Java Object Request Broker (ORB). For details, see “Object Request Broker (ORB)” on page 3.
- Beginning with the 1.1.x version, the Object Discovery Agent (ODA) component of the connector for CORBA uses the IDLJ compiler tool to generate Java proxy class definitions from IDL files. The ODA does not use any other IDL compiler tool. For details, see “IDLJ to Java compiler tool” on page 3.

Users of the adapter for CORBA, version 1.0.x, must recompile their IDL files to regenerate Java proxy class definitions that are compatible with the ODA of version 1.1.x. For details about this migration step, see “IDL file compatibility” on page 51.

- Beginning with the 1.1.x version, the adapter for CORBA is compatible with the IBM ORB Transient Naming Server only. The adapter is not compatible with other naming services. For details, see “IBM ORB Transient Naming Server” on page 4.
- Beginning with the 1.1.x version, the adapter for CORBA does not have a `poa_name` connector-specific configuration property. For details about the valid connector-specific configuration properties, see “Connector-specific properties” on page 22.

Chapter 1. Overview

- “Adapter environment”
- “Terminology” on page 5
- “Architecture of the connector running as a client” on page 6
- “Architecture of the connector running as a server” on page 11
- “Business object requests” on page 14
- “Verb processing” on page 14
- “Custom business object handlers” on page 15

The connector for CORBA (Common Object Request Broker Architecture) is a runtime component of the WebSphere Business Integration adapter for CORBA. The CORBA Adapter includes a connector, message files, configuration tools, and an Object Discovery Agent (ODA). The connector allows the WebSphere integration broker to exchange data between business objects and their corresponding CORBA objects running on a CORBA server. It also allows a CORBA object to submit client requests against the connector, in which case the connector runs as a CORBA server.

Connectors consist 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, CORBA) 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.

All WebSphere business integration adapters operate with an integration broker. The connector for CORBA operates with WebSphere InterChange Server, WebSphere MQ Integrator Broker, or WebSphere Application Server. For more information, see the installation and implementation documentation of your broker.

Adapter environment

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

- “Broker compatibility” on page 2
- “Adapter standards” on page 2
- “Adapter platforms” on page 2
- “Adapter dependencies” on page 2
- “Locale-dependent data” on page 5

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. Version 1.2.x of the adapter for CORBA is supported on the following versions of the adapter framework and with the following integration brokers:

- Adapter framework:
 - WebSphere Business Integration Adapter Framework, version 2.4.0
- Integration brokers:
 - WebSphere InterChange Server, version 4.2.2
 - WebSphere MQ Integrator, version 2.1.0
 - 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 CORBA 2.3.1 specification and as such is compatible with CORBA applications designed to this standard.

Adapter platforms

The adapter runs on the following platforms:

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

Adapter dependencies

The connector for CORBA has the following dependencies, each with unique installation requirements, depending on your adapter platform.

- “JDK software” on page 3

- “Object Request Broker (ORB)”
- “IDLJ to Java compiler tool”
- “JavaC” on page 4
- “IBM ORB Transient Naming Server” on page 4

JDK software

The Java Development Kit (JDK), Version 1.3.1, is a prerequisite of installing the adapter for CORBA.

Windows 2000: WebSphere Business Integration Adapter Framework, Version 2.4.0 provides a separate installation of IBM JDK, Version 1.3.1 SR5. Note that the IBM JDK, Version 1.3.1 SR5 is not installed as part of the WebSphere Business Integration Adapter Framework installation. You must run a separate installation to install the JDK. For details about how to install the JDK from WebSphere Business Integration Adapter Framework, refer to that software package.

Solaris: Install the Sun JDK 1.3.1 that is specified by the WebSphere Business Integration Adapter Framework, Version 2.4.0 installation. The JDK is not installed as part of the WebSphere Business Integration Adapter Framework installation. You must run separate installation software, provided by Sun Microsystems, to install the JDK.

AIX: Install the IBM JDK, Version 1.3.1 SR5. For details about obtaining the installation software, contact IBM Technical Support.

HP-UX: Install the HP JDK 1.3.1 that is specified by the WebSphere Business Integration Adapter Framework, Version 2.4.0 installation. The JDK is not installed as part of the WebSphere Business Integration Adapter Framework installation. You must run separate installation software, provided by HP, to install the JDK.

Object Request Broker (ORB)

The adapter for CORBA assumes an object request broker environment that supports the IBM Java Object Request Broker (ORB).

Windows 2000: The required IBM Java ORB is provided in the IBM JDK 1.3.1 SR5. For instructions about installing the ORB, see the JDK installation package.

Solaris: WebSphere Business Integration Adapter Framework, Version 2.4.0 provides the necessary IBM Java ORB software. The ORB is installed as part of the Adapter Framework installation.

AIX: The required IBM Java ORB is provided in the IBM JDK 1.3.1 SR5. For instructions about installing the ORB, see the JDK installation package.

HP-UX: WebSphere Business Integration Adapter Framework, Version 2.4.0 provides the necessary IBM Java ORB software. The ORB is installed as part of the Adapter Framework installation.

IDLJ to Java compiler tool

The Object Discovery Agent (ODA) component of the connector for CORBA uses the IDLJ compiler tool. This tool is a compiler that the ODA uses to generate the Java proxy class definitions that allow the connector to act as a CORBA client or server.

CORBA structs, interfaces, methods, and other programmatic entities are defined in an IDL (Interface Definition Language) file. The IDLJ compiler tool run by the

ODA converts the CORBA programmatic entities defined in the IDL file into proxy classes. At runtime, the connector creates a proxy object from a proxy class and uses the proxy object to invoke the methods of that object's corresponding CORBA class that is defined in the IDL file.

Users of the adapter for CORBA, version 1.0.x, must recompile their IDL files to regenerate Java proxy class definitions that are compatible with the ODA of version 1.2.x. For details about this migration step, see "IDL file compatibility" on page 51.

Note: Certain application providers of CORBA servers supply the .jar file that defines the proxy classes, thus bypassing the need for the object definitions generated (compiled) by the connector's ODA (using the IDLJ compiler tool). If the CORBA server application provider supplies a .jar file of proxy class definitions, then be sure to copy the file to the *ProductDir*\connectors\CORBA\ext directory, or whatever directory you designate as the directory for storing the proxy class .jar files, where *ProductDir* represents the directory where the connector product is installed. Using Business Object Designer, you specify the appropriate directory name, in the *OutputFileDir* configure agent property. For details about this property and how to set it, see "Configure the agent" on page 53. For details about defining properties in the ODA, see Chapter 5, "Creating and modifying business objects," on page 51.

Windows 2000: The required IDLJ compiler tool is provided in the IBM JDK 1.3.1 SR5. For instructions about installing IDLJ, see the JDK installation package.

Solaris: WebSphere Business Integration Adapter Framework, Version 2.4.0 provides the necessary IDLJ software. The IDLJ is installed as part of the Adapter Framework installation.

AIX: The required IDLJ compiler tool is provided in the IBM JDK 1.3.1 SR5. For instructions about installing IDLJ, see the JDK installation package.

HP-UX: WebSphere Business Integration Adapter Framework, Version 2.4.0 provides the necessary IBM IDLJ software. The IDLJ is installed as part of the Adapter Framework installation.

JavaC

JavaC, the Java programming language compiler, is a prerequisite of the Adapter for CORBA. JavaC is part of the JDK 1.3.1 and as such is installed as a part of any JDK installation. For details about the JDK prerequisite, see "JDK software" on page 3

IBM ORB Transient Naming Server

The IBM ORB Transient Naming Server is a prerequisite of the connector for CORBA. This service provides the necessary CORBA naming services that allow CORBA objects to be named by means of binding a name to an object reference. When running as a client, the connector is able to connect with CORBA servers that use the IBM Java Object Request Broker (ORB) only. When running as a server, CORBA clients that use ORBs of other ORB vendors are able to connect with the connector. For details about the IBM Java ORB, see "Object Request Broker (ORB)" on page 3.

WebSphere Business Integration Adapter Framework, Version 2.4.0 provides the necessary IBM ORB Transient Naming Server software. This software is installed as

part of the Adapter Framework installation. To launch the IBM ORB Transient Naming Server, run one of the following commands, depending on your platform, from the \connectors\CORBA\ directory.

Windows 2000	Solaris, AIX, HP-UX
NamingService.bat	NamingService.sh

This command requires a user-defined port number. For example, the following command, run in a Windows 2000 environment, starts an instance of the IBM ORB Transient Naming Server at port 1100:

```
NamingService.bat 1100
```

Note that to use the required IBM ORB Transient Naming Server software provided by WebSphere Business Integration Adapter Framework, you may need to re-register your existing CORBA services with the service that is launched by the NamingService.bat/NamingService.sh command.

Locale-dependent data

The connector has been internationalized so that it can support delivery of double-byte character sets going into a CORBA 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 runtime 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, verb, 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.
- **Connection object** A special kind of proxy object that is an instance of the connection class. A connection is a reference to an application that can contain state information. For every instance of a connection on the adapter side, there is a corresponding object on the CORBA side. Connections can be instantiated in batches, retrieved at will, sent back to the connection pool, and be re-used by another thread.
- **Connection pool** A repository used to store and retrieve connection objects.

- **CORBA object** The connector interacts with a CORBA server by processing between a business object and a CORBA object. During connector processing, a CORBA object (application) is represented in the connector by a proxy object. A **proxy** is a Java class that represents a CORBA object.
- **Factory** A special kind of proxy object that refers to an application. If the appropriate connector properties are set, the factory object, which is persistent for the life of the connector, can create connections that are placed in the connection pool or can create CORBA objects to be used by CORBA applications. The number of connections created depends on the value specified in the `PoolSize` property.
- **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 CORBA uses the foreign key to specify poolable connection objects.
- **IDLJ** The connector for CORBA supports IBM Java Object Request Broker (ORB), which uses the IDLJ compiler tool. This tool allows Java programs to communicate with CORBA objects by generating the Java proxy classes that the connector requires at runtime to generate proxy objects that in turn invoke CORBA objects. The properties, structures, and methods of a CORBA object are defined in an IDL (Interface Definition Language) file. Using the proxy object class definitions that the IDLJ compiler tool creates through the ODA, the connector can invoke the CORBA methods of an object that are defined in the IDL.
- **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.
- **ORB (Object Request Broker)** A component in the CORBA programming model that acts as the middleware between clients and servers. In the CORBA model, a client can request a service without knowing anything about what servers are attached to the network. The various ORBs receive the requests, forward them to the appropriate servers, and then hand the results back to the client.
- **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 proxy objects or simple attributes.
- **Proxy class** A Java class that represents a CORBA object in the connector. The connector creates a proxy object instance of the proxy class name specified in the business object's ASI.
- **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. It can contain the name of the method to call to process the current request business object.

Architecture of the connector running as a client

The connector can process requests in the following two ways:

- The connector acts as a client that sends business object requests to a CORBA server, as illustrated in Figure 1 on page 7. These requests invoke methods on objects running on an external CORBA server.

- The connector acts as a server that receives requests from external CORBA clients, as illustrated in Figure 3 on page 12. These requests invoke collaborations on the integration broker, which, for example, can update data in an external application.

This section describes the architecture of the CORBA connector when it runs as a client. For details about the CORBA connector running as a server, see “Architecture of the connector running as a server” on page 11.

Request flow

Figure 1 illustrates the request flow when the connector runs as a client. In this scenario, the connector invokes a method on an object that resides on an external CORBA server. The connector acts as a client communicating with the CORBA server through the ORB. The connector communicates with the CORBA server by sending object requests to the server where the CORBA objects reside.

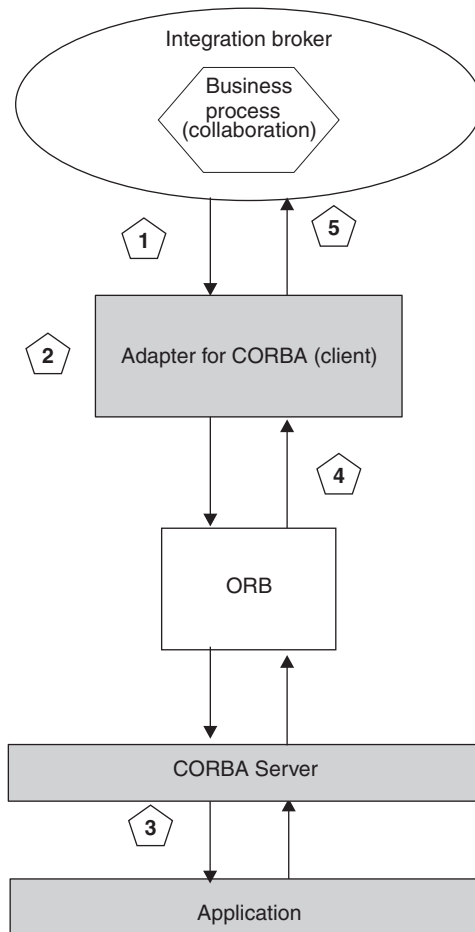


Figure 1. Request process when the connector acts as a client

1. The connector receives a business object request from the integration broker.
2. The connector creates a proxy object instance of the business object. The proxy object instance acts as a representation of the CORBA object to which the connector is sending the request. For details about how the connector creates and processes the proxy object, see “How the connector works as a client” on page 8.

3. The connector processes the proxy object by using it to access the corresponding CORBA object running on CORBA server and write data to the application (object). The connector can also invoke methods on the CORBA object.
4. The connector updates the proxy object by reading, or getting, data from CORBA server object.
5. The connector 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.

How the connector works as a client

This section describes in detail how the different parts of the connector process a business object when the connector runs as a client, as illustrated in Figure 2 on page 9.

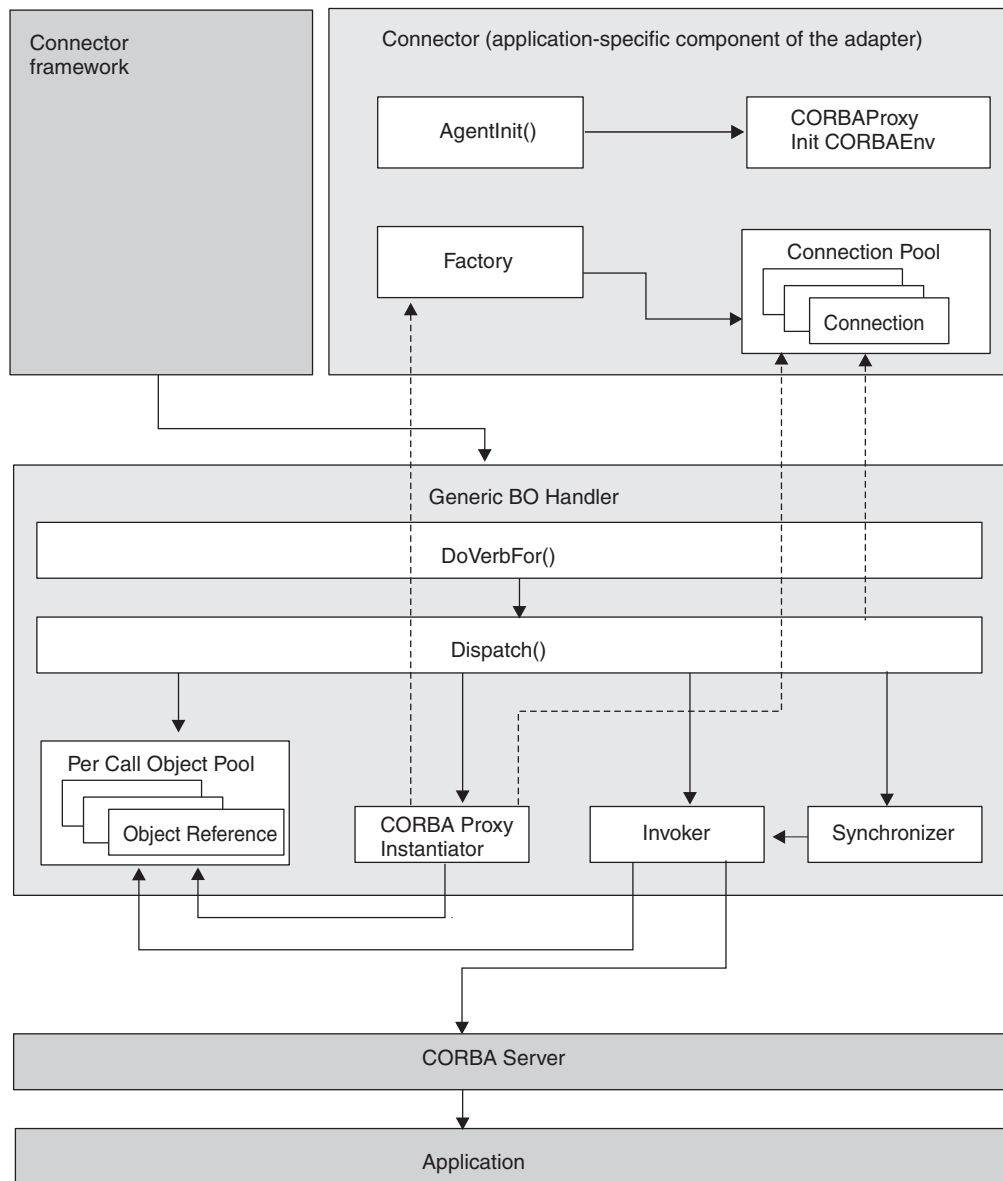


Figure 2. The connector for CORBA running as a client

1. When you first start up the connector and run it as a client, the connector's Agent class performs the following initialization processes:
 - Instantiates the CORBA environment by initializing the ORB.
 - Does one of the following, depending on how the connector properties have been set. For details about the connector properties and how they affect each of the following scenarios, see "Connector-specific properties" on page 22.
 - **Scenario 1:** Creates a factory object instance, which is an object that refers to an application. The factory object, which is persistent for the life of the connector, creates connections that are placed in the connection pool. The number of connections created depends on the value specified in the connector PoolSize property.

- **Scenario 2:** Creates connection objects only that are placed in the connection pool. The number of connections depends on the value specified in the PoolSize property. No factory object is created in this scenario.
 - **Scenario 3:** Creates a factory proxy object against which the business object will call methods (the factory class matches the proxy class ASI of the BO). In this scenario, no connections are created.
2. The integration broker sends a request, in the form of a business object, to the connector.
 3. The connector's BO handler receives the object.
 4. The doVerbFor() method of the BO handler calls the Dispatch() method, which reads the BO ASI to obtain the proxy class name. The Dispatch() method gets the proxy class name and sends it to the CORBA Proxy Instantiator.
 5. The CORBA Proxy Instantiator uses the proxy class name to load the proxy class (qualified using valid Java class notation, ie. xxxxx.myclass) and create a proxy object instance, loading it in the per-call object pool. The CORBA Proxy Instantiator verifies if the object is one of the following:
 - Is it a connection? If so, retrieve it as a connection object from the connection pool.
 - Is it a factory object? If so, retrieve it as a static object from the factory. The CORBA Proxy Instantiator also checks whether or not a factory method is specified in the business object ASI. If so, then it uses the factory method on the factory object.
 6. Dispatch reads through the BO's verb ASI and builds a list of methods. The verb ASI is an ordered list of attribute names. Each attribute represents a method on the proxy object. In other words, the verb ASI is not a list of methods, but a list of attributes, each one having a value that represents a proxy object method.
 7. For each method on the verb ASI list, the InvokeMethods() method of the BO handler calls InvokeMethod() to do one of the following:
 - Call Invoker, if the method is a regular method. If the argument is marked as a foreign key, store it in the per-call object pool. If the attribute is not populated, check the attribute ASI for use_attribute_value. If the use_attribute_value ASI is present, attempt to pull the object from the per-call object pool.
 - Call the Load and Store operations of Synchronizer (the BO handler's object synchronization process) against all attributes on the proxy object. The operation called depends on what is in the verb ASI. LoadFromProxy (Load) and WriteToProxy (Store) are pre-defined functions that you can include in the verb ASI. Their purpose is to synchronize a business object's simple attributes with a CORBA object's public properties.
 - Call Load (LoadFromProxy function) or Store (WriteToProxy function) operations against a single, specific attribute (LoadFromProxy gets the proxy property and sets the BO property to that value; WriteToProxy sets the proxy property with values from the BO).

Note: If the verb ASI is empty, the BO handler will search for a method on the BO with populated parameters and call that. Only one method can have populated parameters. Otherwise, if multiple methods are populated and the verb ASI is empty, then the connector logs an error and returns a FAIL code.

8. For each method of the proxy object, Invoker constructs the parameters and arguments of the method by doing the following:
 - If it encounters a BO type (rather than a simple data type, such as a String) in the attribute, Invoker recursively calls the Dispatch() method on the active BO handler.
 - Dispatch() returns a proxy object that the parent method can use to invoke its method call.
 - The BO handler's synchronization process, called Synchronizer, invokes WriteToProxy to store (set) a value in each property of the CORBA object (proxy object), thus updating data on the CORBA server. The value stored is from the corresponding attribute on the business object that the CORBA object corresponds to.
9. When values are returned from the CORBA server, the LoadFromProxy function loads the data returned from the proxy object onto the BO. (For return parameters, the connector creates the return proxy object and also updates the in / out parameters).
10. The connector returns the business object back to the integration broker.

Architecture of the connector running as a server

In addition to processing requests as a client, the connector can act as a server that receives requests from external CORBA clients. These requests invoke collaborations on the integration broker, which, for example, can update data in an external application.

This section describes the architecture of the connector when it runs as a server. For details about the connector running as a client, see "Architecture of the connector running as a client" on page 6.

Request flow

Figure 3 on page 12 illustrates the request flow when the connector runs as a server. In this scenario, the connector executes a collaboration on behalf of a CORBA client.

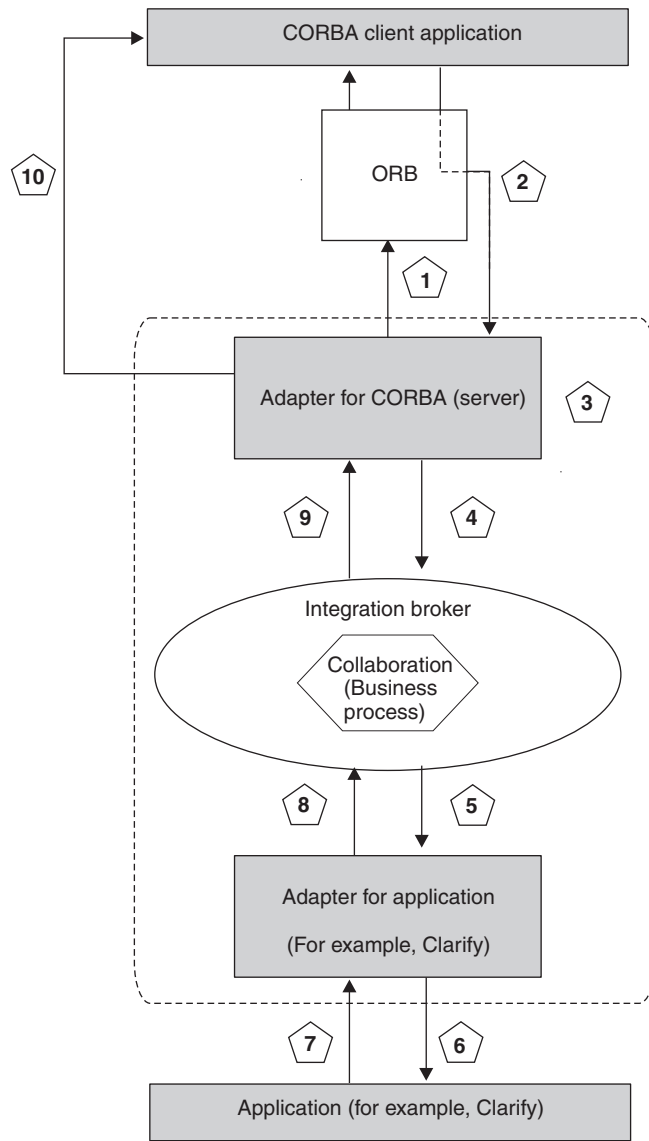


Figure 3. Request process when the connector acts as a server

1. When you first start the connector, all business objects that have been configured as server-side objects are registered by the connector with the ORB. For details about configuring a business object as a server, see Chapter 4, “Understanding business objects,” on page 35.
2. The CORBA client application sends a method call request through the connector against a registered server object.
3. The connector converts the method call to the verb that has been specified in the attribute ASI of the registered CORBA server object. For details about the attribute ASI of a business object, see “Attribute-level ASI” on page 42.
4. The connector invokes the `executeCollaboration()` method of the adapter’s Agent class. Based on the connector’s `BO_COLLAB_MAPPING` property, which associates a business object verb (Step 3) to a particular collaboration, `executeCollaboration()` calls the appropriate collaboration (business process). For details about the `BO_COLLAB_MAPPING` property, see “Configuring the connector” on page 21.

5. The integration broker sends data from the collaboration or business process that was executed to the connector for the external application (that is, the application with which the collaboration is exchanging data; in Figure 3 on page 12, the example of this external application is Clarify).
6. The connector of the external application sends the data to that application.
7. The application returns a message to the connector of the external application (for example, Clarify, as shown in Figure 3 on page 12).
8. The connector of the external application forwards the message to the integration broker.
9. The integration broker forwards the message to the connector acting as a CORBA server (the connector that invoked the `executeCollaboration()` method in Step 4 on page 12).
10. An exception, if it has occurred, is sent to the CORBA client application that initiated the original request (Step 2 on page 12).

How the connector works running as a server

When running as a server, the connector performs business object processing in a different manner than when running as a client. This section describes how the different parts of the connector, running as a server, process business objects, as illustrated in Figure 4.

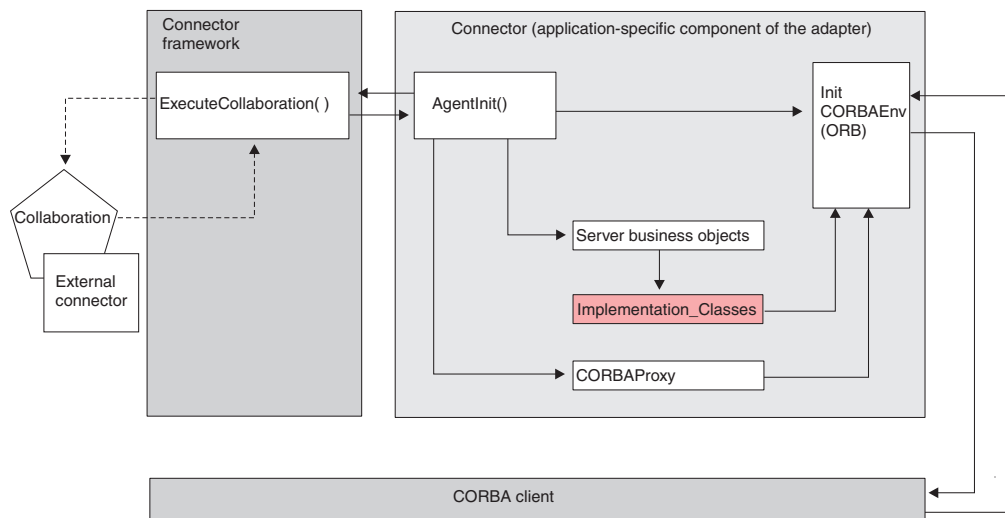


Figure 4. The connector for CORBA running as a server

1. When you start the connector, its Agent class performs the following initialization processes:
 - Instantiates the CORBA environment by initializing the ORB.
 - Passes information to the implementation classes about the business objects that will be processed as server objects. To determine if a business object is a server object, the connector verifies that the BO ASI includes the attribute `object_type=CorbaImplObject`.
 - Registers with the ORB the implementation classes that correspond to the business objects. You specify the implementation class of a given business object in the `implementation_class` ASI. Class definitions are stored in the

- .jar file generated by the ODA when you create the business objects. For details, see Chapter 5, “Creating and modifying business objects,” on page 51.
2. The Agent class sets up the factory, the instance of the factory proxy object, or the connection objects, depending on how the connector properties have been set. For details about connector properties, see “Connector-specific properties” on page 22.
 3. The external CORBA client sends to the CORBA server business object in the connector a request to invoke a collaboration. The request, sent via the ORB, is in the form of input arguments of a method.
 4. As described in step 3 on page 12 and step 4 on page 12:
 - The connector creates a business object using the argument values of the method.
 - The connector then sets the verb on the business object, as specified in the attribute ASI of the CORBA server business object.
 - Next, the connector calls the `executeCollab()` method of the connector’s Agent class.
 - The `executeCollab()` method looks for the collaboration specified in the `BO_COLLAB_MAPPING` property, a connector property that maps the business object verb to a collaboration. If the collaboration is found, the connector executes the collaboration. If the collaboration is not found, an exception is raised. (For return parameters, the connector creates the return proxy object and also updates the in / out parameters).
 5. The data resulting from collaboration processing is then passed back to the CORBA client application via the ORB.

Business object requests

Business object requests are processed when the integration broker sends a business object to the connector. The only requirement of the business object is that it must map to the corresponding CORBA object that the proxy object will represent. The proxy class is a Java class that represents a CORBA object in the connector. At runtime, the connector creates a proxy object instance of the proxy class name specified in the business object’s ASI.

Verb processing

Note: This section pertains to the connector when it acts as a client. The verb-processing issues described here do not apply when the connector runs as a server.

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, the BO handler searches for a method with populated parameters and calls that. 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 63.

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 51.

Note: You can specify two pre-defined functions in the verb ASI: `LoadFromProxy` and `WriteToProxy`. Their purpose is to synchronize a business object’s simple attributes to a CORBA object’s public properties.

Custom business object handlers

When you create a business object, you can override the default BO handler by specifying the `CBOH` keyword in the BO verb ASI. At connector runtime, the `doVerbFor()` method retrieves the business object ASI. If it detects the `CBOH` keyword, `doVerbFor()` invokes the custom BO handler.

The connector supports custom BO handlers on parent-level business objects only. For details about creating a custom BO handler, see the *Connector Development Guide*.

Chapter 2. Installing the adapter

- “Overview of installation tasks”
- “Connector file structure” on page 18
- “Post-installation tasks” on page 19

Overview of installation tasks

To install the adapter for CORBA, 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 detail, 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 CORBA supports, see “Broker compatibility” on page 2.

For details about how to install the broker, see the appropriate implementation documentation of the broker you are using.

Install the adapter for CORBA 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>

Windows and AIX installation

When installing the connector on a Windows or AIX platform, note that Installer will ask you to specify the directory pathname where the IBM JDK 1.3.1 SR5 is located (as explained in “JDK software” on page 3, this software is a prerequisite for installing the connector for CORBA).

Make sure that you have the directory information readily available so that you can provide it during installation. Otherwise, you will have to prematurely cancel out of Installer while you locate the JDK directory on your machine, and then re-run Installer from the beginning.

The adapter provides the following script that you can run to obtain the pathname of the JDK home directory:

- `\connectors\CORBA\BIA_CORBAEnv.bat` (Windows)
- `\connectors\CORBA\BIA_CORBAEnv.sh` (Unix)

Connector file structure

The adapter installation copies the standard files associated with the connector into your system.

The utility installs the connector into the *ProductDir*\connectors\CORBA directory, and adds a shortcut for the connector to the Start menu. Note that *ProductDir* represents the directory where the product 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 <i>ProductDir</i>	Description
\connectors\CORBA\BIA_CORBA.jar	Contains classes used by the CORBA connector only
\connectors\CORBA\start_CORBA.bat	The startup script for the generic connector (Windows)
\connectors\CORBA\start_CORBA.sh	The startup script for the generic connector (Unix)
\connectors\CORBA\ext\	A directory where the ODA-generated .jar files can be saved. If you save to this directory, specify the directory in the startup script (start_CORBA.bat or start_CORBA.sh).
\connectors\CORBA\NamingService.bat	The IBM Transient Naming Server startup file (Windows).
\connectors\CORBA\NamingService.sh	The IBM Transient Naming Server startup file (Unix).
\connectors\messages\BIA_CORBAConnector.txt	Message file for the connector
\connectors\CORBA\BIA_CORBAEnv.bat	Script file that tracks the pathname of the JDK home directory.
\connectors\CORBA\BIA_CORBAEnv.sh	Script file that tracks the pathname of the JDK home directory (Unix).
\ODA\CORBA\BIA_CORBAODA.jar	The CORBA ODA
\ODA\CORBA\start_CORBAODA.bat	The ODA startup file (Windows)
\ODA\CORBA\BIA_CORBAODA.sh	The ODA start up file (Unix)
\ODA\messages\BIA_CORBAODAAgent_de_DE.txt	Message file for the ODA (German text strings)
\ODA\messages\BIA_CORBAODAAgent_en_US.txt	Message file for the ODA (US English text strings)
\ODA\messages\BIA_CORBAODAAgent_es_ES.txt	Message file for the ODA (Spanish text strings)
\ODA\messages\BIA_CORBAODAAgent_fr_FR.txt	Message file for the ODA (French text strings)
\ODA\messages\BIA_CORBAODAAgent_it_IT.txt	Message file for the ODA (Italian text strings)
\ODA\messages\BIA_CORBAODAAgent_ja_JP.txt	Message file for the ODA (Japanese text strings)
\ODA\messages\BIA_CORBAODAAgent_ko_KR.txt	Message file for the ODA (Korean text strings)
\ODA\messages\BIA_CORBAODAAgent_pt_BR.txt	Message file for the ODA (Portuguese - Brazil text strings)
\ODA\messages\BIA_CORBAODAAgent_zh_CN.txt	Message file for the ODA (Simplified Chinese text strings)
\ODA\messages\BIA_CORBAODAAgent_zh_TW.txt	Message file for the ODA (Traditional Chinese text strings)
\repository\CORBA\BIA_CN_CORBA.txt	Repository definition for the connector. The default name is BIA_CN_CORBA.txt.

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

Post-installation tasks

After installation and before startup, you must configure the adapter. For details, see Chapter 3, “Configuring the adapter,” on page 21.

Chapter 3. Configuring the adapter

- “Post-installation tasks” on page 19
- “Configuring the connector”
- “Creating multiple connector instances” on page 31
- “Starting the connector” on page 32
- “Stopping the connector” on page 33
- “Using log and trace files” on page 33

Overview of configuration tasks

After installation and before startup, you must configure components as described in this section.

Configure the connector

Configuring the connector includes setting up and configuring the connector. For details, see “Configuring the connector.”

Configure the business objects

You create and configure business objects through Business Object Designer. You can create a business object definition using an ODA (Object Discovery Agent), through Business Object Designer. 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 work with it interactively. You use Business Object Designer to edit an already existing business object.

For details business objects, see Chapter 4, “Understanding business objects,” on page 35.

For details about using the ODA, see Chapter 5, “Creating and modifying business objects,” on page 51.

Configuring the connector

Connectors have two types of configuration properties: standard configuration properties and adapter-specific configuration properties. You must set the values of these properties using Connector Configurator before running the adapter. For further information, see Appendix B, “Connector Configurator,” on page 87.

A connector obtains its configuration values at startup. During a runtime 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 after a change. 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.

Standard connector properties

Standard connector configuration properties provide information that all adapters use. See Appendix A, “Standard configuration properties for connectors,” on page 69 for documentation of these properties.

Although the following properties are listed in Appendix A, “Standard configuration properties for connectors,” on page 69, the connector for CORBA does not use them:

- DuplicateEvent Elimination
- PollEndTime
- PollFrequency
- PollStartTime

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 runtime. 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 87.

Note that all the connector-specific properties are optional, unless otherwise noted, in that you can choose to set them based on your specific connector configuration requirements. For example, when running the connector as a client, do you want the connector to create both factory objects and connections, only a factory object, or only connections? The configuration you choose determines which properties you must set.

Table 2 lists the connector-specific configuration properties for the connector, along with their descriptions and possible values. The + character indicates the entry’s position in the property hierarchy. See the sections that follow for details about the properties, including the hierarchy of these properties illustrated in Figure 5 on page 24.

Table 2. Connector-specific configuration properties

Name	Possible values	Default value
+ Factory	<i>None. This is simply a category in the hierarchy.</i>	None
+ + FactoryClass	<i>The class name</i>	None
+ + FactoryMethod	<i>The method name</i>	None
+ + + Arguments	<i>Any encrypted or non-encrypted strings</i>	
+ + FactoryInitializer	<i>The method name of the initializer</i>	None
+ + + ior_file_name	<i>The directory and file name of the IOR file.</i>	None
+ + + name	<i>The name of the object used for getting the factory; for example: HelloServerServerObject</i>	None

Table 2. Connector-specific configuration properties (continued)

Name	Possible values	Default value
+ ConnectionPool	<i>None. This is simply a category in the hierarchy.</i>	None
+ + ConnectionClass	<i>Class name</i>	None
+ + ConnectionInitializer	<i>Method name of the initializer</i>	None
+ + + Arguments	<i>Any encrypted or non-encrypted strings</i>	None
+ + PoolSize	<i>Any integer</i>	0
+ UseNamingContext	True, False	None
+ ClientOnly	True, False	None
+ BO_COLLAB_MAPPING	<i>The business object and verb mapped to the collaboration. The structure is: <businessObject.verb> mapped to <collaborationName></i>	None
+ CORBAServerName	<i>A valid CORBA server name; for example, CORBAAdapter.</i>	None
+ BO_CONNECTION_PROP	<i>The name of the business object mapped to the ior_file_name or name; for example: Customer mapped to ior_file_name=c:\psrserver.ior; name=;</i>	None
+ ORBInitialHost	<i>The host name of the CORBA name server. Must be the IBM ORB Transient Naming Server.</i>	None
+ ORBInitialPort	<i>The port number of the CORBA name server. Must be the IBM ORB Transient Naming Server.</i>	None

Figure 5 on page 24 illustrates the hierarchical relationship of the connector-specific properties.

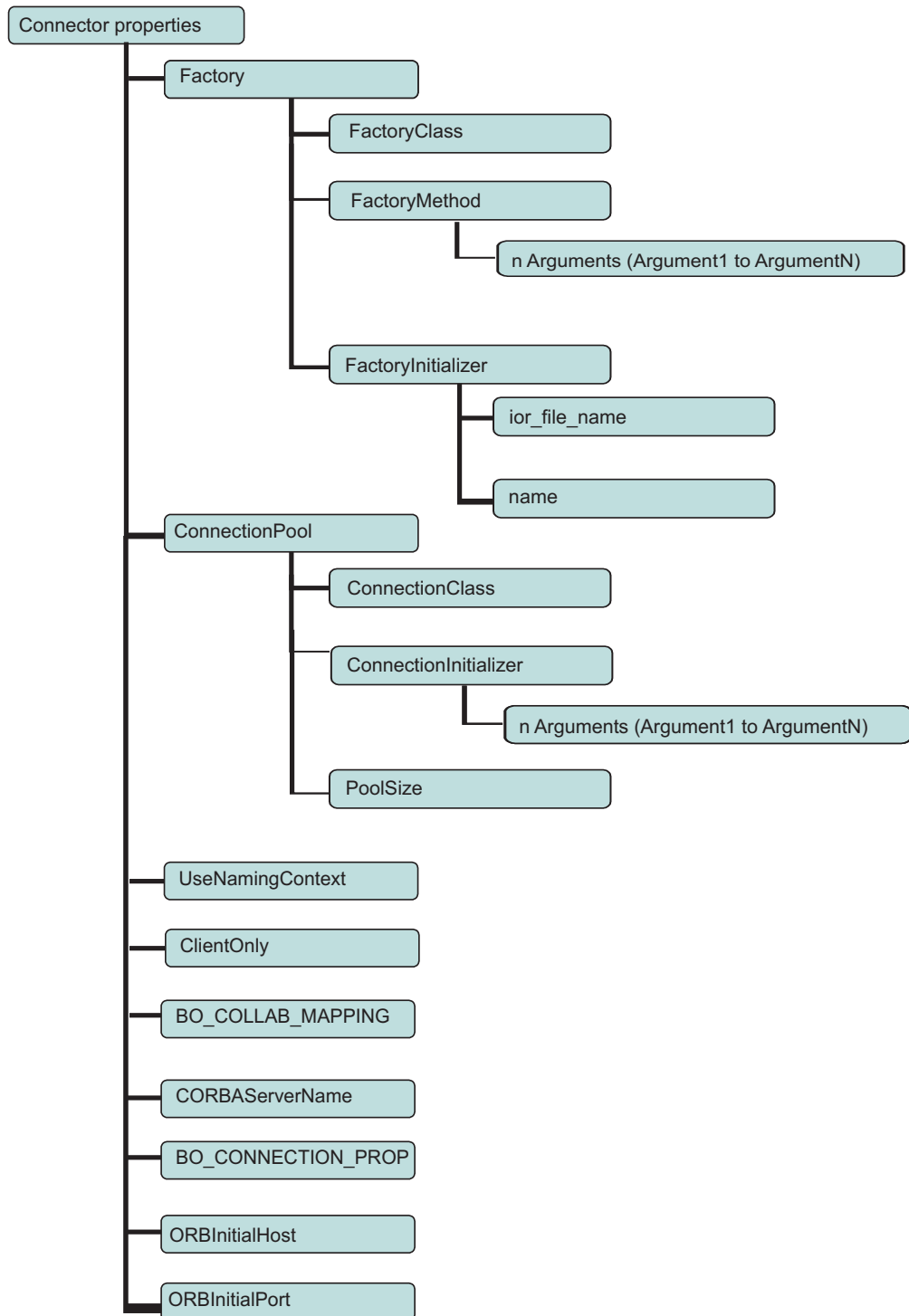


Figure 5. Hierarchy of connector-specific properties

Factory

A hierarchical property that represents the category of the Factory class information.

FactoryClass

The name of the factory class.

- If you specify a `FactoryClass` *and* a `ConnectionClass`, the connector instantiates a factory proxy object and connections (**Scenario 1** in step 1 on page 9).
- If you do not specify a `FactoryClass`, then specify a `ConnectionClass`, in which case a connection pool of the specified connection class and size is created when you initialize the connector (**Scenario 2** in step 1 on page 9).
- If you specify a `FactoryClass` only, the connector will instantiate a factory proxy object (**Scenario 3** in step 1 on page 9) and will not use connections.

You can use this property when the connector runs as a client.

FactoryMethod

A hierarchical property that represents the name of the `FactoryMethod` on the `FactoryClass`. If you specify a `FactoryMethod`, `ConnectionInitializer` is called on the connection object it gets from the `Factory` method (after that object is created).

You can use this property when the connector runs as a client.

Arguments

The parameters of the `FactoryMethod` must be arguments (`Argument1`, `Argument2`, and so on) on the `FactoryClass`, listed in proper sequential order. The property names are `Argument1`, `Argument2`, and so on, for as many parameters as the method takes. The value of each argument is any encrypted or non-encrypted string

You can use this property when the connector runs as a client.

FactoryInitializer

A hierarchical property that represents the way the `Factory` class must be initialized.

You can use this property when the connector runs as a client.

ior_file_name

The name of the `ior_file` for the `Factory` object. The connector uses this property to instantiate the `Factory` object. You can specify this property, or `name`, or both.

You can use this property when the connector runs as a client.

name

The name of the object used for getting the `Factory`. This property is used to instantiate the `Factory` object and is required if you have not specified a value for `ior_file_name`. This property is also required if the adapter uses a naming context (the `useNamingContext` property is set to `true`).

You can use this property when the connector runs as a client.

ConnectionPool

A hierarchical property that represents the category of the `Connection` class information.

ConnectionClass

The name of the poolable connection class.

- If you specify a `ConnectionClass` *and* a `FactoryClass`, the connector instantiates a factory proxy object and connections (**Scenario 1** in step 1 on page 9).
- If you specify a `ConnectionClass`, but not a `FactoryClass`, then, when the connector is initialized, a connection pool instance is created for storing connections (**Scenario 2** in step 1 on page 9).

The size of the pool (number of connections) is based on the value you specify in the PoolSize property.

If you use connection pooling on a multi-use server (one instance of a server object can be re-used to create a connection), then you have to set up a factory and factory method call to create the connection pool. In this case, each BO handler thread pulls a discrete connection from the pool to be used during processing.

You can use this property when the connector runs as a client.

ConnectionInitializer

The name of the poolable ConnectionClass initializer method.

If you specify a FactoryMethod, ConnectionInitializer is called on the connection object it gets from the FactoryMethod (after that object is created).

You can use this property when the connector runs as a client.

Arguments

The parameters of the ConnectionInitializer must be arguments (Argument1, Argument2, and so on) on the ConnectionClass, listed in proper sequential order. The property names are Argument1, Argument2, and so on, for as many parameters as the initializer takes. The value of each argument is any encrypted or non-encrypted string

You can use this property when the connector runs as a client.

PoolSize

Determines the size of the connection pool. The default value is 0.

This property is required if the ConnectionClass is specified. Use this property when the connector runs as a client.

UseNamingContext

A boolean true/false flag that indicates whether or not the connector uses the CORBA naming service to look up the object with the specified value for name. For details about the naming services, see "IBM ORB Transient Naming Server" on page 4.

This is a required property when the connector runs as either a client or a server.

ClientOnly

A boolean true/false flag that indicates whether or not the connector runs as a CORBA client. If the flag is set to false, then the connector runs as both a CORBA server and a client.

This is a required property.

BO_COLLAB_MAPPING

If the connector runs as a CORBA server (the ClientOnly property is set to false), this hierarchical property is used to map a business object and its verb (*businessObject.verb*) to a collaboration. The collaboration executes when an external CORBA client places a method call against a server object. After receiving the method call, the connector converts the method to the appropriate verb (as specified in the object's attribute ASI). Then, using the verb-collaboration mapping

specified in the `BO_COLLAB_MAPPING` property, the server object calls the `executeCollaboration()` method, which in turn executes the appropriate collaboration for that verb.

This property is required only when the connector processes business objects that map to CORBA servers. Use this property when the connector runs only as a server.

CORBAServerName

The name that is used when registering the connector as a CORBA server.

This property is required only when the connector processes business objects that map to CORBA servers. Use this property when the connector runs only as a server.

BO_CONNECTION_PROP

The information required for mapping business objects to connection information. You set this property when the connector runs as a client. The information includes the name of the supported client business object, the `ior_file_name` and the name. For example, if the connector has two supported business objects, `CORBACustomer` and `CORBAAccount`, the `BO_CONNECTION_PROP` setting is as follows:

```
CORBACustomer=ior_file_name=<fileName>;name=<name>
```

```
CORBAAccount=ior_file_name=<fileName>;name=<name>
```

The connector uses this property in one of the following ways, depending on the values provided in the property setting:

- If `ior_file_name` is populated, then the connector uses this value to look up the business object.
- If `name` is populated with a value and the `UseNamingContext` property is set to `true`, then the connector uses the naming context to look up the CORBA object of the specified name value.

If neither of these property setting scenarios is true (in other words `ior_file_name` and `name` are blank), then the connector logs an error in the log file stating that the necessary connection information is missing for the business object. In addition, the BO handler is not initialized for the given business object. For details about error logging, see Chapter 6, “Troubleshooting and error handling,” on page 63.

This property is required only when the connector processes business objects that run as CORBA clients.

ORBInitialHost

The host name of the IBM ORB Transient Name Server. Whether the connector runs as a client or server, it must use the naming services of the IBM ORB Transient Naming Server. For details about the naming server, see “IBM ORB Transient Naming Server” on page 4.

This property is required. It is used by the connector to connect to a CORBA server, or to act as a server itself. Even if the name server is running on the same host as the connector, you must specify this.

ORBInitialPort

The port number of the IBM ORB Transient Name Server. Whether the connector runs as a client or server, it must use the naming services of the IBM ORB Transient Naming Server. For details about the naming server, see “IBM ORB Transient Naming Server” on page 4.

This property is required. It is used by the connector to connect to a CORBA server, or to act as a server itself. Even if the name server is running on the default port (900), you must specify this.

Sample configuration scenarios

This section provides examples of the following configuration scenarios:

- “Configuring the connector to run as a client using a `FactoryClass` and `ConnectionClass`”
- “Configuring the connector to run as a simple client without using a `FactoryClass` and `ConnectionClass`” on page 29
- “Configuring the connector to run as a server” on page 30

Configuring the connector to run as a client using a `FactoryClass` and `ConnectionClass`

When you run the connector as a client that makes calls to an external CORBA server, determine in advance which service—Transient Name Server (TNS) or Interoperable Object Reference (IOR)—the server is using, so as to set connector properties accordingly. With TNS, the adapter calls go through the network, whereas with IOR, the calls are file-based in that the network information required to locate the server is stored in an IOR file that the adapter reads.

The following sample IDL file (`HelloFactory.idl`) is used to generate business objects required by the connector when it runs as a client, using the `FactoryClass` and `ConnectionClass`, against a server that uses TNS. For an example of the connector running as a simple client *without* the `FactoryClass` and `ConnectionClass`, see “Configuring the connector to run as a simple client without using a `FactoryClass` and `ConnectionClass`” on page 29.

```
module HelloAppFactory
{
    interface HelloApp
    {
        string sayHello();
    };

    interface HelloFactory
    {
        HelloApp getHello();
        HelloApp getHelloUsingName(in string name);
    };
};
```

The sample code contains the following elements:

- `interface HelloApp` refers to the `ConnectionClass`.
- `interface HelloFactory` refers to the `FactoryClass`.
- `HelloApp getHello()` refers to the `FactoryMethod`, which returns an instance of the `HelloApp ConnectionClass`.

Table 3 on page 29 provides sample configuration settings of some of the connector-specific properties for the `HelloFactory.idl` scenario. For a complete list of connector-specific properties, see Table 2 on page 22.

Table 3. Connector-specific properties for HelloFactory.idl scenario (Server configuration)

Connector-specific property	Value
UseNamingContext	true
ClientOnly	true
FactoryClass	HelloAppFactory.HelloFactory
FactoryInitializer > name	HelloApp_HelloFactoryServerNaming
FactoryMethod	getHello
ConnectionClass	HelloAppFactory.Hello
PoolSize	10
BO_CONNECTION_PROP	Client_HelloAppFactory_HelloFactory
Client_HelloAppFactory_HelloFactory	name=HelloApp_HelloFactoryServerNaming

If the connector is running as a client and making calls to a CORBA server that uses the IOR service rather than TNS, note the following requirements:

- Specify the IOR file name (`ior_file_name= ;`) in the `BO_CONNECTION_PROP > Client_HelloAppFactory_HelloFactory` property.
 - `Client_HelloAppFactory_HelloFactory` is the name of the client business object used to connect the CORBA client to the server.
 - IOR file name (`ior_file_name= ;`) is used to look up the business object.
- Set the `UseNamingContext` property to `false`.

Configuring the connector to run as a simple client without using a FactoryClass and ConnectionClass

The following sample IDL file for module `corbaadapter_sample` defines an interface (`Hello`) that is used to generate business objects required by the connector when it runs as a client with TNS but *not* using the `FactoryClass` and `ConnectionClass`. For an example of the connector running as a client that uses the `FactoryClass` and `ConnectionClass`, see “Configuring the connector to run as a client using a `FactoryClass` and `ConnectionClass`” on page 28.

```
//code for simple client and server config
module corbaadapter_sample
{
    typedef sequence<long> LongSeq;
    typedef sequence<string> StringSeq;

    interface Hello
    {
        //simple type tests
        string simpleIn(in LongSeq in_long_val, in double in_amount,
            in boolean in_istrue, in string in_firstNm)
            raises (ProcessingFailureException);
        StringSeq simpleOut(out LongSeq out_long_val,
            out double out_amount, out boolean out_istrue,
            out string out_firstNm) raises (ProcessingFailureException);
    };
};
```

Table 4 on page 30 provides sample configuration settings for the connector-specific properties you configure when using module `corbaadapter_sample`, which defines interface `Hello`. For a complete list of connector-specific properties, see Table 2 on page 22.

Table 4. Connector-specific properties for interface Hello scenario (simple client without a FactoryClass and ConnectionClass)

Connector-specific property	Value
UseNamingContext	true
ClientOnly	true

Notice in this scenario that ClientOnly is set to true and that the FactoryClass, FactoryInitializer, FactoryMethod, ConnectionClass, and ConnectionPool properties are not used. These properties are relevant only when configuring the connector as a client that uses the FactoryClass and ConnectionClass.

Configuring the connector to run as a server

When you configure the connector to run as a server, it can receive requests from external CORBA clients. These requests invoke collaborations on the integration broker.

The following sample IDL file for module corbaadapter_sample defines an interface (interface Hello) that is exposed to CORBA clients when the connector is configured with the property settings listed in Table 5. When the connector runs as a server, it uses the business objects created from this IDL file to service requests from CORBA clients.

```
//code for simple client and server config
module corbaadapter_sample
{
    typedef sequence<long> LongSeq;
    typedef sequence<string> StringSeq;

    interface Hello
    {
        //simple type tests
        string simpleIn(in LongSeq in_long_val, n double in_amount,
            in boolean in_istrue ,in string in_firstNm) raises
            (ProcessingFailureException);
        StringSeq simpleOut(out LongSeq out_long_val,
            out double out_amount,out boolean out_istrue ,
            out string out_firstNm) raises (ProcessingFailureException);
    };
};
```

Table 5 provides sample configuration settings for the connector-specific properties you set when using module corbaadapter_sample, which defines interface Hello. For a complete list of connector-specific properties, see Table 2 on page 22.

Table 5. Connector-specific properties for interface Hello scenario (Server configuration)

Connector-specific property	Value
UseNamingContext	true
ClientOnly	false
BO_COLLAB_MAPPING	Server_corbaadapter_sample_Hello.Create

Notice that in this scenario the FactoryClass, FactoryInitializer, FactoryMethod, ConnectionClass, and ConnectionPool properties are not used, as these properties are only relevant when configuring the connector as a client that uses the FactoryClass and ConnectionClass.

The BO_COLLAB_MAPPING property represents the server object (formatted as *businessObjectName.verb*) that the connector, acting as a server, maps to a

collaboration. The collaboration runs when an external CORBA client places a method call against an instance of this server object, which creates the interface defined in module `corbaadapter_sample`.

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 31.
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*.

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 6 shows.

Table 6. 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 [-cconfigFile ]
```

- 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.
- On Windows systems, you can configure the connector to start as a Windows service. In this case, the connector starts when the Windows system boots (for an Auto service) or when you start the service through the Windows Services window (for a Manual service).

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.
- On Windows systems, you can configure the connector to start as a Windows service. In this case, the connector stops when the Windows system shuts down.

Using log and trace files

The adapter 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 63.

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 87.

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 53. The ODA trace levels are the same as the connector trace levels, defined in “Tracing” on page 66.

Chapter 4. Understanding business objects

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

The chapter contains the following sections:

- “Defining metadata”
- “Connector business object structure” on page 36
- “Mapping attributes: CORBA, Java, and business object” on page 44
- “Sample business object properties” on page 45
- “Generating business objects” on page 49

Defining metadata

The connector for CORBA is metadata-driven. In the WebSphere business integration system, metadata is defined as application-specific information that describes a CORBA application object’s data structures. The metadata is used to construct business object definitions, which the connector uses at runtime 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 are created in Business Object Designer, without the assistance of the ODA. To modify an existing object, use Business Object Designer directly (the ODA cannot be used to modify an existing business object).

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. Therefore, it is important that the structure of the business object exactly mirror the structure defined for the corresponding CORBA object or the adapter 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 object in CORBA and then export the changes to the file system repository for input into the ODA.

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

Connector business object structure

The connector processes two kinds of business objects, both of which are generated by the ODA:

- Business objects used by CORBA server-side components when the connector runs as a client
- Business objects used by CORBA client-side components when the connector runs as a server

Note that if an IDL component is intended both as a client and a server-side object, the ODA has to generate two separate business objects, one for client-side processing and one for server-side processing. In this case, make sure to run the ODA twice against the same IDL component: the first time, run it with the ODA `CORBAServerImpl` property set to `true` to generate a server-side business object; then, run the ODA again with the same property set to `false`. For details about this property, see “Configure the agent” on page 53.

This section describes the key concepts related to the structure of business objects processed by the CORBA connector.

Attributes

For each attribute present in a CORBA class defined in the IDL file, a corresponding business object attribute is generated by the ODA. The IDL file is used by the ODA to compile proxy object definitions.

If an attribute in the CORBA class is not a simple attribute, and instead is a struct (Figure 6 on page 37), union (Figure 9 on page 38), or sequence (Figure 8 on page 37), then the BO attribute maps to a child object whose definition matches the corresponding constructed type (construct) in the CORBA object. The CORBA enum construct (Figure 10 on page 38) maps to a simple attribute, rather than to a child object attribute.

The primary constructs of CORBA are described in Table 7 and illustrated in Figure 6 on page 37 through Figure 9 on page 38. (Note that the connector does not support the constant construct.)

For a complete list of the mapping between CORBA constructs and business objects, see “Mapping attributes: CORBA, Java, and business object” on page 44.

Table 7. CORBA Constructs

CORBA construct	Description
struct	An object that holds business data, as illustrated in Figure 6 on page 37
interface	An object that holds a list of business operations (methods), as illustrated in Figure 7 on page 37
sequence	An object that holds a list of structs or simple data types, as illustrated in Figure 8 on page 37. A CORBA sequence maps to a cardinality <i>n</i> business object, as does an array.
union	A collection of structs or simple data types, as illustrated in Figure 9 on page 38. Note that only one attribute within a union can have a value at a given time.
enum	An object that contains a list of sequential or enumerated identifiers, as illustrated in Figure 10 on page 38.

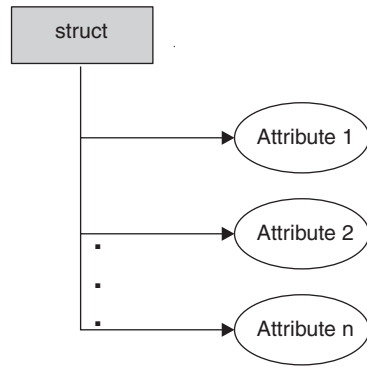


Figure 6. CORBA construct: struct

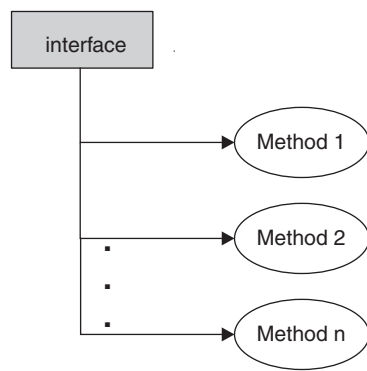


Figure 7. CORBA construct: interface

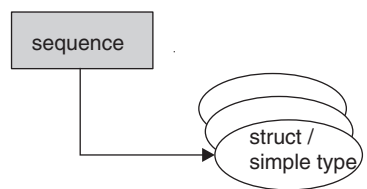


Figure 8. CORBA construct: sequence

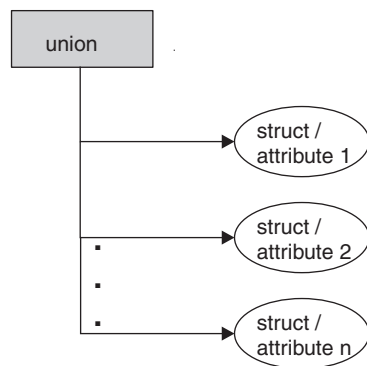


Figure 9. CORBA construct: union

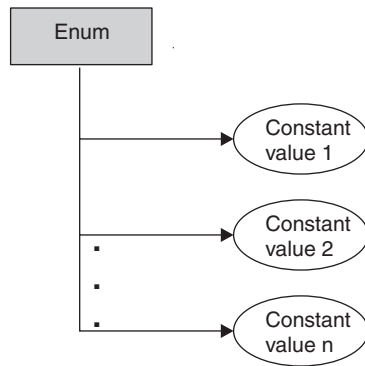


Figure 10. CORBA construct: enum

Some business object attributes, instead of containing data, point to child business objects or arrays of child business objects that contain the data for these objects. Keys relate the data between the parent record and child records.

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 same as that of 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.

Methods

For each method defined in the CORBA IDL file, an attribute is created in the business object. The attribute type is a child BO containing attributes that represent method parameters. The attributes of the child BO appear in the exact same order as the parameters of the CORBA method. The child BO also has a `Return_Value` attribute that represents the result of the CORBA method call. These attributes (of the child BO) can be simple type or object type, depending on the type of the method parameter or return value.

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 can be specified for the overall business object as well as for each business object attribute.

Business object-level ASI

Object-level ASI provides fundamental information about the nature of a business object and the objects it contains. The required ASI for a business object depends on whether you are generating the object for the connector running as a server or as a client.

Note: Application specific information is used for business objects that represent methods, method parameters, and method return values. For details about business object attributes created for methods of CORBA objects, see “Methods” on page 38.

Table 8 describes the business object-level ASI of business objects that are processed as client objects when the connector for CORBA runs as a client.

Table 8. Object-level ASI for client objects

Object-level ASI	Description
proxy_class=<nameOfProxy>	The name of the proxy class that the business object represents. Use this ASI to map a proxy class to a business object. You must specify this using valid Java Package notation (for example, java.lang.Vector).
factory_method=<Name Of Factory Method>	The name of the method of the Factory class used to instantiate the specified proxy_class. This release supports only a method that does not take any arguments.
object_type=<leave blank or set to zero-length string>	If the CORBAServerImpl property of the ODA agent (see Table 15 on page 54) is set to false, meaning that the ODA is configured to generate client-side objects, this ASI should not be specified or should be set to a zero-length String.
implementation_class=<leave blank or set to zero-length string>	If the CORBAServerImpl property of the ODA agent (see Table 15 on page 54) is set to false, meaning that the ODA is configured to generate client-side objects, this ASI should not be specified or should be set to a zero-length String.

Table 9 describes the business object-level ASI of business objects that are processed as server objects when the connector acts as a server against which external CORBA client objects can make method calls. Note that the ODA generates server-side business objects when you set the CORBAServerImpl property of the ODA agent to true. For details about this property, see Table 15 on page 54.

Table 9. Object-level ASI for server objects

Object-level ASI	Description
proxy_class=<nameOfProxy>	The name of the proxy class that the business object represents. Use this ASI to map a proxy class to a business object. This must be specified using Java Package notation (for example, java.lang.Vector).
object_type=CorbaImplObject	If the connector runs as a server and the BO is intended for CORBA clients to invoke method calls against the BO, set this ASI to CorbaImplObject.

Table 9. Object-level ASI for server objects (continued)

Object-level ASI	Description
<code>implementation_class=<Name Of Implementation Class></code>	<p>The name of the implementation class that corresponds to the server business object you are creating in the ODA. The implementation class name is always:</p> <pre>com.ibm.adapters.corbaadapter.impl.<ClassName>Impl</pre> <p>where <ClassName> is the name of the original class being implemented. The classes are stored in an output .jar file and saved to the following folder:</p> <pre>com\ibm\adapters\corbaadapter\impl</pre>

Figure 11 illustrates the object level ASI for a sample business object acting as a server object.

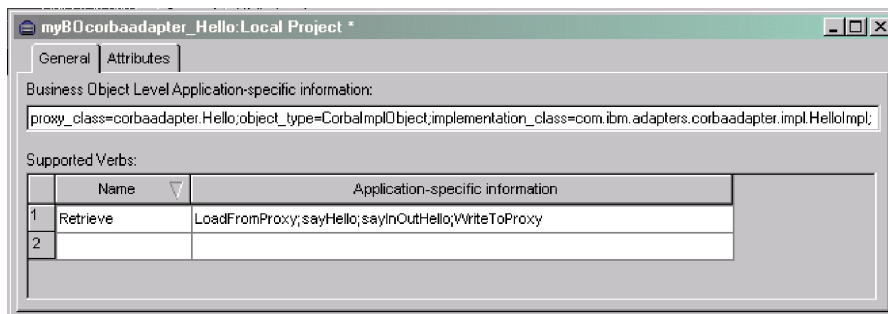


Figure 11. Sample business object-level ASI of a server-side object

Verb ASI

When the connector runs as a client, every client-side business object that it processes contains a verb. The verb describes how the data in the business object should be handled by the receiving application.

Note: Server-side objects, processed when the connector runs as a server, do not have a verb ASI.

The verb ASI contains a sequence of attribute names, each of which contains a method for the generic business object handler to call. Typically, the method to be invoked belongs to the object itself (versus belonging to the parent), in which case you specify the method in the object's verb ASI. For example, an object that has the method `IncrementCounter` would require that you specify that method in the corresponding business 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 by prefixing the method name with the `PARENT` tag.

For example, Figure 12 on page 41 illustrates a business object hierarchy whereby `ContactDetails` is a child object of `Contact`, which itself is a child of

PSRCustomerAccount.

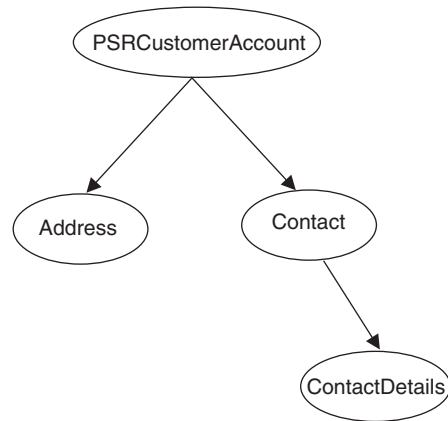


Figure 12. Business object hierarchy and verb ASI

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. Furthermore, a parent business object cannot invoke a child's method.

The connector developer determines the CORBA operations assigned to the verb. Although you can add other verbs manually using Business Object Designer, the supported verbs include:

- Create
- Delete
- Retrieve
- Update

The following keywords can be used in the verb ASI sequence of attribute names:

Table 10. Keywords allowed in verb ASI

Keyword	Description
LoadFromProxy= <attributeName>	Loads the specified non-method attribute value from the proxy object.
WriteToProxy = <attributeName>	Writes the non-method attribute value from the business object into the proxy object.
LoadFromProxy (no attribute name)	Loads all the non-method attributes on the current BO from proxy object.
WriteToProxy (no attribute name)	Writes all the non-method attributes on the current BO to the proxy object.

Table 10. Keywords allowed in verb ASI (continued)

Keyword	Description
CBOH=<custom BO handler className>	The class name of a custom BO handler, in cases where the generic BO handler is not used. For information about custom BO handlers, see “Custom business object handlers” on page 15.

For a given object, you can specify the four supported verbs (Create, Delete, Retrieve, and Update) and assign as actions of each verb n plus two methods, where n equals the number of methods in the corresponding CORBA interface. The two additional methods are those supported by the connector, LoadFromProxy and WriteToProxy, as defined in Table 10 on page 41.

Attribute-level ASI

The attribute-level ASI of a business object can be for complex attributes, which contain child objects, and simple attributes. For a complex attribute, the ASI varies, depending on whether the contained child is a property (non-method) or a method of the object. The mapping of all the attribute-types in the original CORBA IDL file to the business object is defined in Table 14 on page 44.

Table 11 describes the ASI for simple attributes.

Table 11. Attribute-level ASI for attributes that contain simple attributes

Attribute ASI	Description
Name	Specifies the business object field name.
Type	Specifies the business object field type.
MaxLength	255 by default
IsKey	Each business object must have at least one key attribute, which you specify by setting the key property to true for an attribute.
IsForeignKey	Set to true if you want the connector to add the value to the per-call object pool.
IsRequired	Set to false.
AppSpecInfo	<p>Holds the original Java type. This attribute is formatted as follows:</p> <pre>property=<propertyName>; type=<typeName></pre> <p>property is the name of the CORBA object attribute. Use this ASI to capture the original CORBA object attribute name.</p> <p>type is the name of the CORBA simple attribute type. Use this attribute to capture the original CORBA type name.</p>
DefaultValue	Specifies a default value that the connector uses for a simple attribute in the inbound business object if the attribute is not set and is a required attribute.

Table 12 describes the ASI for complex, non-method attributes. These attributes contain non-method child objects, such as properties of a class in the original CORBA IDL file.

Table 12. Attribute-level ASI for attributes that contain non-method child objects

Attribute	Description
Name	Specifies the business object field name.
Type	Specifies the business object field type.

Table 12. Attribute-level ASI for attributes that contain non-method child objects (continued)

Attribute	Description
MaxLength	255 by default
IsKey	Each business object must have at least one key attribute, which you specify by setting the key property to true for an attribute.
IsForeignKey	Set to true if you want the connector to add the value to the per-call object pool.
IsRequired	Set to false.
AppSpecInfo	<p>Holds the original Java type. This attribute is formatted as follows:</p> <pre>type=<typeName>; use_attribute_value=<BOName.AttributeName>; property=<propertyName>; proxy_class=<proxyClassName>; enumeration_class=<enumerationClassName>; inout=<true or false>; union=true; union_key=<unionKeyName>;</pre> <p>type is the name of the proxy class when you are referring to an object.</p> <p>use_attribute_value is set to <BOName.AttributeName>. If you specify a value for this element, the connector pulls the value from the per-call object pool at runtime.</p> <p>property is the name of the CORBA object attribute. Use this ASI to capture the original CORBA object attribute name.</p> <p>proxy_class is optional. Use only if the attribute is of type in / out.</p> <p>enumeration_class is the CORBA enumeration class to which this non-method attribute corresponds. Use this optional element only if the attribute maps to a CORBA enumeration construct.</p> <p>inout is set to true or false. Use this optional element only if the attribute is of type in / out as a method parameter.</p> <p>union should be set to true. Optional if the child BO maps to a CORBA union construct.</p> <p>union_key is the mapping key to each method that returns a value in the union. Use this optional element only if the BO maps to a CORBA union construct.</p> <p>For details about CORBA constructs and structures, see “Attributes” on page 36</p>
DefaultValue	Specifies a default value that the connector uses for a simple attribute in the inbound business object if the attribute is not set and is a required attribute.

Table 13 describes the ASI for complex attributes containing child objects that are methods.

Table 13. Attribute-level ASI for attributes that contain method child objects

Attribute	Description
Name	Specifies the business object field name.
Type	Specifies the business object field type.
Relationship	If the child is a container attribute, this is set to Containment.
IsKey	Not used.
IsForeignKey	Set to false.

Table 13. Attribute-level ASI for attributes that contain method child objects (continued)

Attribute	Description
Is Required	Set to false.
AppSpecInfo	<p>Holds the original CORBA application method name. This attribute is formatted as follows:</p> <pre>method_name=<methodName>; verb=<verbName>;</pre> <p>method_name is the name of the method call placed to the external CORBA server, when the connector runs as a client.</p> <p>verb is the verb that the connector sets on the business object before invoking the collaboration, when the connector runs as a server. This verb corresponds to the method call from the external CORBA client. The method call is a request to execute a collaboration. Use only if the BO acts as a CORBA server object that is receiving method calls from an external CORBA client object. For a list of valid verbs, see “Verb ASI” on page 40</p>
Cardinality	Set to N if the type represents an array or vector, otherwise set to 1.

Mapping attributes: CORBA, Java, and business object

This section provides a list of the main CORBA IDL constructs and the corresponding Java constructs and 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 Java proxy object.

For details about CORBA constructs, see “Attributes” on page 36.

For details about business object ASI, see “Application-specific information” on page 38.

Table 14. Object mapping: CORBA, JAVA, and business object

CORBA IDL construct	Java construct	Business object	ASI
module	package	(does not apply)	(does not apply)
interface (non-abstract)	Signature interface and an operations interface, helper class, holder class	BO	proxy_class=<fully qualified CORBA class name>
interface (abstract)	Signature interface, helper class, holder class	BO	proxy_class=<fully qualified CORBA interface name>
boolean	boolean	Boolean	type=boolean
char, wchar	char	String	type=char
octet	byte	String	type=byte
string, wstring	java.lang.String	String	type=String
short, unsigned short	short	Integer	type=short
long, unsigned long	int	Integer	type=int
long, long unsigned long long	long	Integer	type=long

Table 14. Object mapping: CORBA, JAVA, and business object (continued)

CORBA IDL construct	Java construct	Business object	ASI
float	float	Float	type=float
double	double	Double	type=double
fixed	java.math.BigDecimal	String	type=BigDecimal
enum	class	String	property=<propertyName>;type=String; enumeration_class=<JavaEnumClassName>
struct, union	class	BO	proxy_class=<fully qualified class name>
sequence, array	array	Child BO with multiple cardinality	proxy_class=<fully qualified class name>
exception	class	(does not apply)	(does not apply)
readonly attribute	accessor method	Child BO	method=<method name>
readwrite attribute	accessor and modifier methods	Child BO	method=<method name>
operation	method	Child BO	method=<method name>

Sample business object properties

This section provides an example of a WebSphere business integration business object. The corresponding CORBA class and Java proxy class are also provided to illustrate the mapping across the three constructs. Note that business objects inherit their names from the matching CORBA application objects.

The samples provided in this section are as follows:

- “Sample IDL file”
- “Sample Java code generated by IDLJ” on page 46
- “Sample business objects for Java classes” on page 48
- “Sample BO handler method calls” on page 48

Sample IDL file

The following sample code is a portion of a larger IDL file. The portion provided here illustrates definitions for the CORBAAccount struct and for the Hello interface which uses that class for method argument types.

```
# Sample IDL File
#
#
module corbaadapter
{
.
.
.
    struct CORBAAccount
    {
        short                accessCustomerNumber;
        AccountStatusEnum    accountStatus;
        string                acctSecurity;
        string                companyNm;
        long                 custAcctID;
        string                disconnectReasonCd;
        string                firstNm;
        string                lastNm;
    }
}
```

```

        char                middleInitial;
        CORBASicCodeUnion   sicCode;
        CORBAAddressSeq     addresses;
        LongSeq             custAcctChildrenIds;
        StringSeq           nameList;
        ShortSeq            accountList;
        BooleanSeq          flagList;
        CharSeq             initialList;
        FloatSeq            amountList;
        DoubleSeq           doubleAmtList;
    };

    interface Hello
    {
        CORBAAccount sayHello(in CORBAAccount test, inout double amount);

        CORBAAccount sayInOutHello(inout CORBAAccount test,
        inout string name,
        in long id);
    };
};

```

Sample Java code generated by IDLJ

The following examples illustrate Java code generated by the IDLJ compiler tool from the code in “Sample IDL file” on page 45.

- “Sample Java code: CORBAAccount class”
- “Sample Java code: HelloOperations class” on page 47

Sample Java code: CORBAAccount class

The following sample code is a portion of the Java code generated by the IDLJ compiler tool for the CORBAAccount struct defined in “Sample IDL file” on page 45.

```

package corbaadapter;

/**
 * <ul>
 * <li> <b>IDL Source</b>      "d:/corba adapter/sample/hello.idl"
 * <li> <b>IDL Name</b>       "::corbaadapter::CORBAAccount"
 * <li> <b>Repository Id</b>  IDL:corbaadapter/CORBAAccount:1.0
 * </ul>
 * <b>IDL definition:</b>
 * <pre>
 * struct CORBAAccount {
 *     ...
 * };
 * </pre>
 */
public final class CORBAAccount implements org.omg.CORBA.portable.IDLEntity {

    public short accessCustomerNumber;

    public corbaadapter.AccountStatusEnum accountStatus;

    public java.lang.String acctSecurity;

    public java.lang.String companyNm;

    public int custAcctID;

    public java.lang.String disconnectReasonCd;

    public java.lang.String firstNm;

```



```

public java.lang.String lastNm;

public char middleInitial;

public corbaadapter.CORBASicCodeUnion sicCode;

public corbaadapter.CORBAAddress[] addresses;

public int[] custAcctChildrenIds;

public java.lang.String[] nameList;

public short[] accountList;

public boolean[] flagList;

public char[] initialList;

public float[] amountList;

public double[] doubleAmtList;

public CORBAAccount () {
}

.
.
.
}

```

Sample Java code: HelloOperations class

The following sample code is the Java class generated by the IDLJ compiler tool for the Hello interface defined in "Sample IDL file" on page 45.

```

package corbaadapter;

/**
 * <ul>
 * <li> <b>IDL Source</b>      "d:/corba adapter/sample/hello.idl"
 * <li> <b>IDL Name</b>       ::corbaadapter::Hello
 * <li> <b>Repository Id</b>  IDL:corbaadapter/Hello:1.0
 * </ul>
 * <b>IDL definition:</b>
 * <pre>
 * interface Hello {
 * ...
 * };
 * </pre>
 */
public interface HelloOperations {
    /**
     * <pre>
     * corbaadapter.CORBAAccount sayHello (in corbaadapter.CORBAAccount test,
     *                                     inout double amount);
     * </pre>
     */
    public corbaadapter.CORBAAccount sayHello (corbaadapter.CORBAAccount test,
                                                org.omg.CORBA.DoubleHolder amount);

    /**
     * <pre>
     * corbaadapter.CORBAAccount sayInOutHello (inout corbaadapter.
     *                                           CORBAAccount test,inout string name, in long id);
     * </pre>
     */

```

```

*/
public corbaadapter.CORBAAccount sayInOutHello
    (corbaadapter.CORBAAccountHolder test,org.omg.CORBA.StringHolder
    name, int id);

```

Sample business objects for Java classes

The following sample screens illustrate the business object structure, as viewed in Business Object Designer, for the Java classes defined in the examples in “Sample Java code generated by IDLJ” on page 46.

Figure 13 illustrates the business object structure for the CORBAAccount class.

General		Attributes								
	Pos	Name	Type	Key	Foreign	Required	Card	Maximum Length	Default	App Spec Info
1	1	accessCustomerNumber	Integer	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>				property=accessCu
2	2	accountStatus	String	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>		255		property=accountSt
3	3	acctSecurity	String	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>		255		property=acctSecur
4	4	companyNm	String	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>		255		property=company
5	5	custAcctID	Integer	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>				property=custAcctI
6	6	disconnectReasonCd	String	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>		255		property=disconnec
7	7	firstNm	String	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>		255		property=firstNm,ty
8	8	lastNm	String	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>		255		property=lastNm,typ
9	9	middleInitial	String	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>		255		property=middleIniti
10	10	<input type="checkbox"/> sicCode	SC_BOCorbaadapter_COR	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	1			property=sicCode,t
11	11	<input type="checkbox"/> addresses	SC_BOCorbaadapter_COR	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	n			property=adresse
12	12	<input type="checkbox"/> custAcctChildrenIds	SC_BOCORBA_Integer	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	n			property=custAcct
13	13	<input type="checkbox"/> nameList	SC_BOCORBA_String	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	n			property=nameList,t
14	14	<input type="checkbox"/> accountList	SC_BOCORBA_Short	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	n			property=accountLi
15	15	<input type="checkbox"/> flagList	SC_BOCORBA_Boolean	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	n			property=flagList,ty
16	16	<input type="checkbox"/> initialList	SC_BOCORBA_Char	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	n			property=initialList,t
17	17	<input type="checkbox"/> amountList	SC_BOCORBA_Float	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	n			property=amountLis
18	18	<input type="checkbox"/> doubleAmtList	SC_BOCORBA_Double	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	n			property=doubleAm
19	19	ObjectEventId	String							
20	20			<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>		255		

Figure 13. Business object structure for CORBAAccount class

Sample BO handler method calls

For the CORBA objects defined in “Sample IDL file” on page 45, the connector BO handler could make the following method calls.

```

//Initialize ORB

ORB orb = ORB.init(args, orbProps);
System.out.println("ORB initialized");
byte[] helloId = "HelloServerObject".getBytes();
Hello helloRef = HelloHelper.bind(orb, "/CORBAServer", helloId);
// Call the Hello server object and print results
CORBAAccount customer = new CORBAAccount();
customer.accessCustomerNumber = 0;
    customer.accountStatus = AccountStatusEnum.asPENDING;
customer.acctSecurity = "check";
customer.companyNm = "Hello";
    customer.custAcctID = 100;
customer.disconnectReasonCd = "Reason";
customer.firstNm = "Name check";
    customer.lastNm = "Last Name";
customer.middleInitial = 'D';

```

```
CORBASicCodeUnion sicCodeUnion = new CORBASicCodeUnion();
CORBASicCode sicCode = new CORBASicCode();
sicCode.description = "Description";
sicCode.sicCd = "1000";
sicCode.stdCdInd = 'N';
sicCode.subAcctInd = 'S';
sicCodeUnion.value(sicCode);
customer.sicCode = sicCodeUnion;
customer.addresses = new CORBAAddress[0];
customer.custAcctChildrenIds = new int[0];
double value = 123;
DoubleHolder dHolder = new DoubleHolder(value);
customer = helloRef.sayHello(customer, dHolder);
```

Generating business objects

Each time an event occurs during run time, a CORBA 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 CORBA application.

Note: If the object model in the CORBA 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 CORBA application sends, the connector is not able 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 51.

Chapter 5. Creating and modifying business objects

This chapter describes the Object Discovery Agent (ODA) for CORBA, and how to use it to generate business object definitions for the IBM WebSphere Business Integration adapter for CORBA.

This chapter contains the following sections:

- “Overview of the ODA for CORBA”
- “IDL file compatibility”
- “Generating business object definitions” on page 52
- “Specifying business object information” on page 57
- “Uploading business object files” on page 62

Overview of the ODA for CORBA

An ODA (Object Discovery Agent) 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 Object Discovery Agent (ODA) for CORBA generates business object definitions from metadata contained in IDL 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 87.

IDL file compatibility

Users of the adapter for CORBA, version 1.0.x, who are migrating to version 1.2.x, must recompile their IDL files to generate proxy class files that are compatible with the ODA component of version 1.2.x. There are two options for doing this:

- **Option 1:** Regenerate the necessary executable files using the ODA, as described in “Generating business object definitions” on page 52
- **Option 2:** Keep the existing business objects but regenerate the Java executables manually through the IBM IDLJ Java compiler tool (`idlj` or `idlj.exe`). To do this, follow these steps:
 1. Recompile the IDL files using IBM IDLJ Java compiler. This compiler is available from the bin directory of the IBM JDK directory for the Window 2000 and AIX platforms. For the Solaris and HP-UX platforms, the compiler is available at the following directory:

```
<adapter runtime directory>/jre/ibm_bin
```

Where *<adapter runtime directory>* is the directory where the adapter runtime files are stored.

If the Java files are generated for the adapter running as a server, invoke the IBM IDLJ Java compiler using the `-fserverTIE` option.

- If the Java files are generated for the adapter running as a client, invoke the IBM IDLJ Java compiler using the `-fclient` option
2. Compile the Java files using the Java compiler available in the JDK, version 1.3.1.
 3. Package the Java executable files (files with a `.class` extension) that you generated in Step 2, into a suitable `.jar` file.

Generating business object definitions

This section describes how to use the CORBA 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 (that is, the IDL files) resides, using the `start_CORBAODA.bat` (Windows) or `start_CORBAODA.sh` (Unix) start file. This file contains start parameters, including the paths to certain required CORBA and connector `.jar` files. These `.jar` files must also be accessible from the machine on which you are running the ODA.

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

To start the ODA, run this command:

- **Windows:** `start_CORBAODA`
- **Unix:** `start_CORBAODA.sh`

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

You must first select the ODA agent.

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 (from the `start_CORBAODA` script) in the **Located agents** list and click **Next**. (You may have to click **Find Agents** if the

desired agent is not listed.)

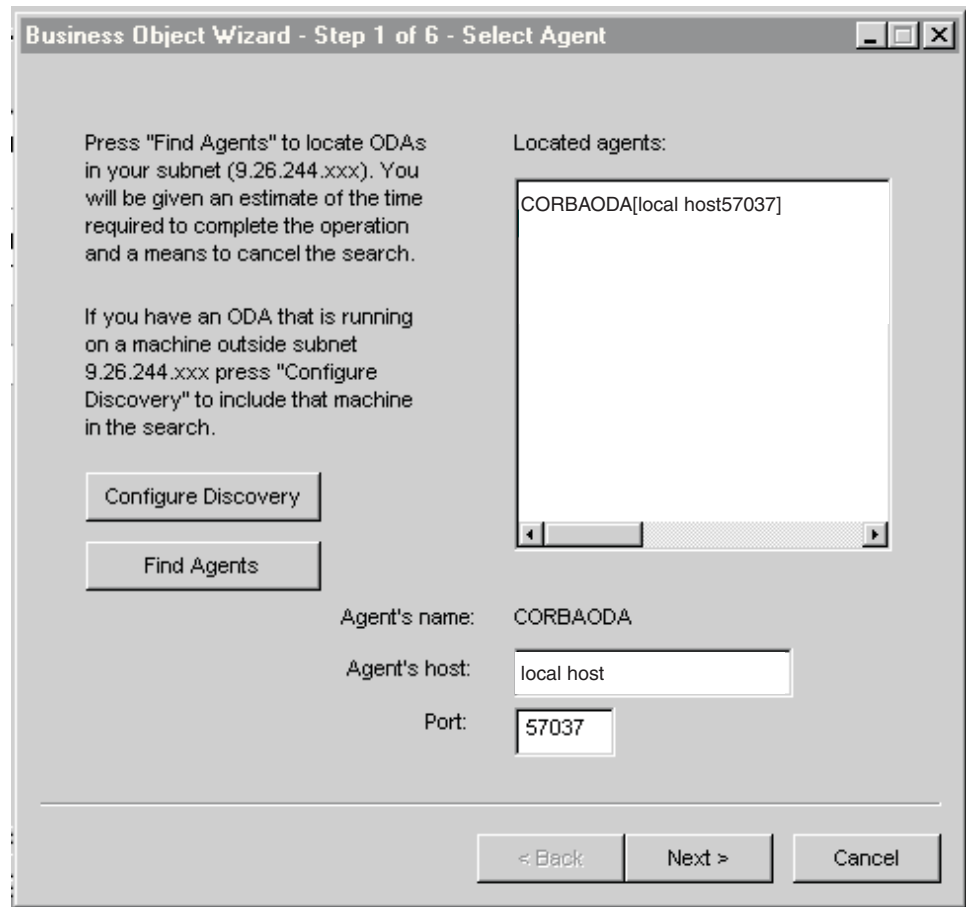


Figure 14. Select Agent screen

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 15 on page 54 illustrates this screen with sample values.

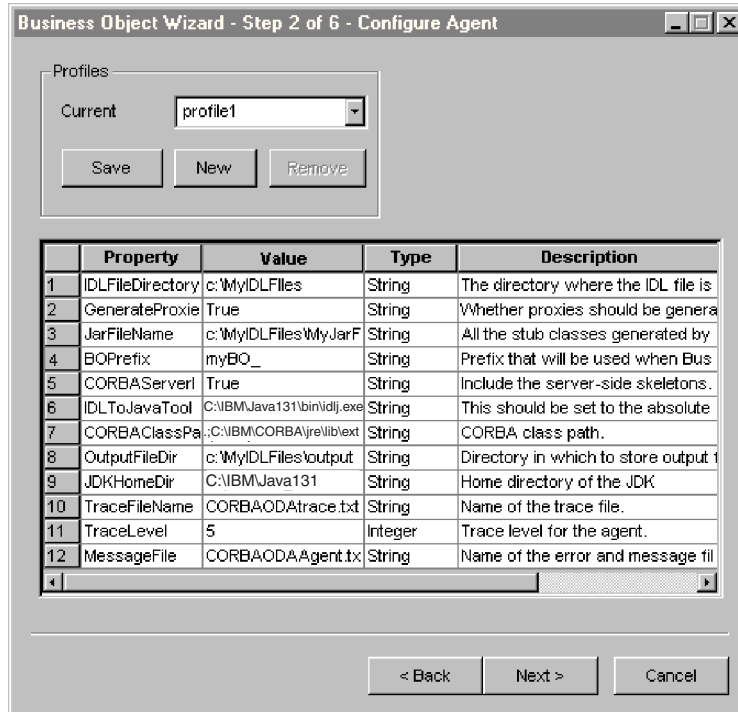


Figure 15. Configure Agent screen

The properties you set on this screen are described in Table 15. You can save all the values you enter on this screen to a profile. Instead of retyping the property data next time you run the ODA, you simply select a profile from the drop-down menu and re-use the saved values. You can save multiple profiles, each with a different set of specified values.

Table 15. Configure Agent properties

Property name	Default value	Type	Description
IDLFileDirectory	None	String	(required) The directory where the IDL files are located. All the IDL files that define the CORBA interface must be placed in this directory.
GenerateProxies	True	Boolean	(required) When set to true, generates the required proxy object classes. When set to false, the adapter uses the .jar file of a CORBA application vendor instead of generating the .jar file of proxy object classes. Make sure to specify the CORBA application vendor's file name in the JCLASSES setting in the start_CORBA.bat (Windows) or start_CORBA.sh (Unix) file. If you are creating server-side objects for a connector that runs as a server, this property is ignored and the ODA generates server implementation classes. This allows you to use the proxy class definitions provided in a .jar file of a CORBA application vendor, instead of generating your own.

Table 15. Configure Agent properties (continued)

Property name	Default value	Type	Description
JarFileName	None	String	The name of the .jar file in which the classes generated by the ODA will be stored. If you specify a file name only (no path), the ODA uses the value you specify in OutputFileDir as the directory in which to output this file. If you specify the absolute path (directory <i>and</i> .jar file name) here, the ODA ignores the value you specify in OutputFileDir. This property is required when the GenerateProxies property is set to true.
BOPrefix	None	String	The prefix that the ODA will add to the names of the business objects it generates. In Figure 15 on page 54, all business objects that the ODA generates, will have names that start with myBO_.
CORBAServerImpl	False	Boolean	(required) When set to true, the ODA generates server-side business objects and implementation class definitions for server-side processing (to be used when the connector runs as a server). If you also want to generate client-side business objects for the same CORBA IDL objects, run the ODA again with this property set to false.
IDLToJavaTool	None	String	The absolute path of the IDLJ compiler tool.
CORBAClassPath	None	String	(optional) The semicolon delimited (Windows) or colon-delimited (Unix) string that contains the pathname of the external CORBA infrastructure class files. These .jar files are required to successfully compile the proxy class definitions. The value of this property is temporarily appended, whenever you run the ODA, to the class path used by the ODA. Set this value explicitly to include all JAR files or directories needed to compile CORBA classes, including any generic CORBA classes such as those defined in ibmorb.jar (the jar file required by the IBM Java ORB).
OutputFileDir	None	String	(required) Directory in which to store all output files generated by the ODA. If you set the GenerateProxies property to false, the ODA does not create a JAR file (JarFileName property) in which the ODA-generated classes are stored. Instead, the classes are saved as separate output files in the directory you specify here. This is a working directory, so as a precaution, make sure each time you run the ODA that the directory you specify here does not already contain any files.
JDKHomeDir	None	String	(required) The directory in which the JDK is installed on this machine.
TraceFileName	None	String	The name of the trace message file; for example, CORBAODATrace.txt.
TraceLevel	5	Integer	(required) The tracing level (from 0 to 5) for the Agent. For details about tracing levels, see “Tracing” on page 66.
MessageFile	None	String	(required) The name of the message file that contains all the messages displayed by the ODA. For CORBA, the name of this file is BIA_CORBAODAAgent.txt. If you do not correctly specify the name of the message file, the ODA will run without messages.

1. Use the **New** and **Save** buttons in the Profiles group box any time you want the ODA to create a new profile. When you use the ODA again, you can select an existing profile.
2. Type the value of each property, as defined in Table 15 on page 54.

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

Select a business object

The *Business Object Wizard - Step 3 of 6 - Select Source* screen appears, as illustrated in Figure 16. The screen lists the interfaces and structs that have been defined in the CORBA IDL file. Use this screen to select any number of CORBA entities for which the ODA will generate business object definitions. A high level parent entity is always an interface or a struct. A sub object of the high-level parent can be an interface, struct, union, enum, or sequence. A descendent union, enum, or sequence is automatically generated as a business object when you select any of its interface or struct ancestors (parent or higher level).

Note: A descendent interface or struct is only generated if you explicitly select it. In other words, it is not automatically generated simply because an ancestor has been selected.

To determine which CORBA objects listed on this screen are child objects of a high-level object, refer to the original IDL file. You can also simply select all the CORBA objects listed on this screen and generate their corresponding business objects. The resulting business objects will reflect the parent-child relationships.

For details about the CORBA structures for which you can create business objects, see Table 7 on page 36.

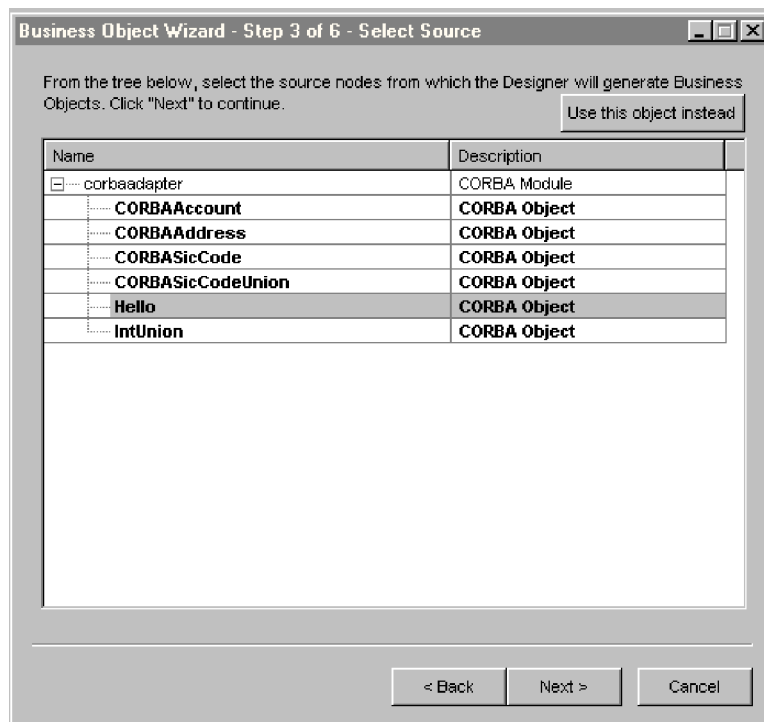


Figure 16. Select Source screen

1. If necessary, expand a CORBA module to see a list of sub objects.
2. Select the CORBA object(s) you want to use. In Figure 16 on page 56, the Hello object is selected
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.

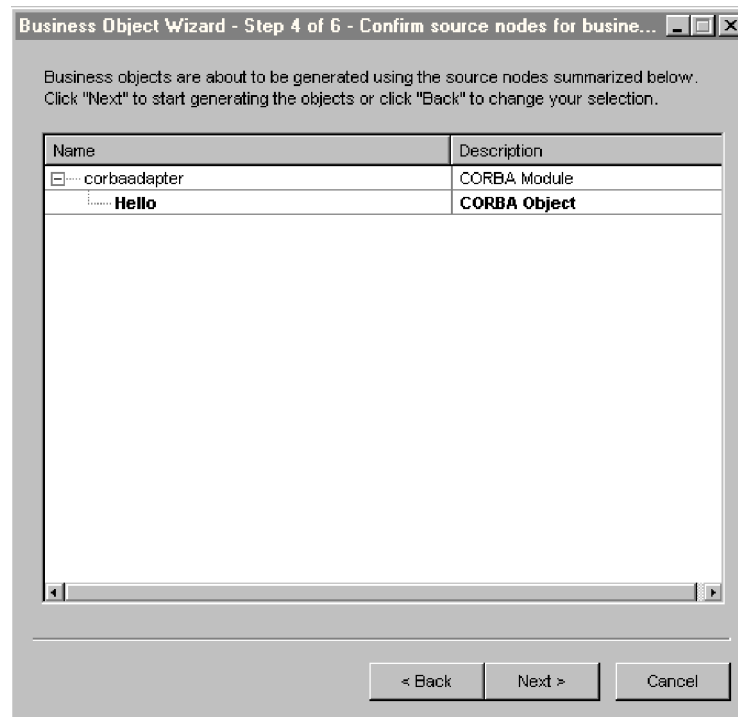


Figure 17. Confirm source node screen

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.

Specifying business object information

After you create a business object, you can specify the verbs that are valid for the object, the method sequence of a given verb on the object, the business object-level ASI, and the attribute-level ASI. This section describes how to specify this information, using the ODA with Business Object Designer. For a detailed description of these categories of information and what they mean for business object structure in the CORBA connector, see Chapter 4, “Understanding business objects,” on page 35.

Selecting verbs

In Business Object Designer, the first screen that appears when you finish creating a business object and then open it in a separate window is the BO Properties -

Select Verbs for component screen. Figure 18 illustrates this screen for the Hello business object created in Figure 16 on page 56 and Figure 17 on page 57.

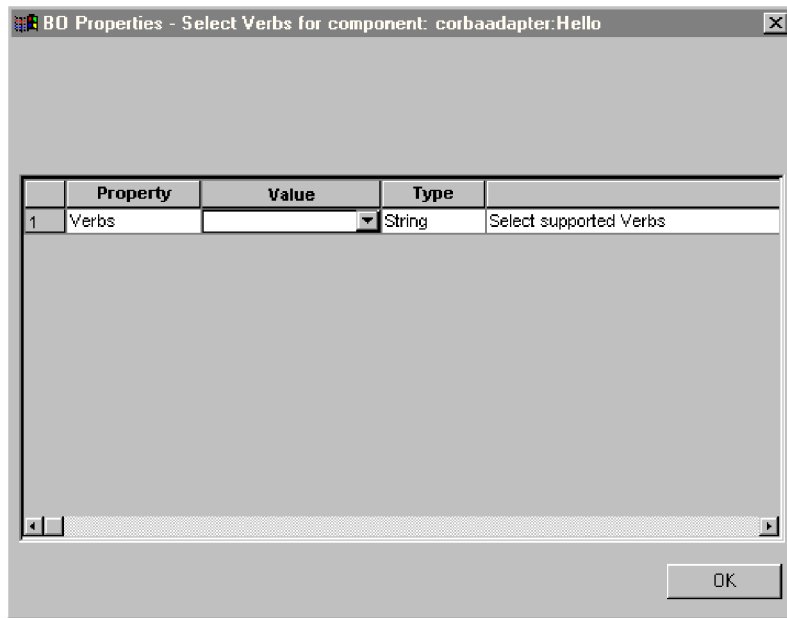
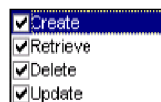


Figure 18. Select verb for component screen

On this screen you specify the verbs that the business objects supports. The ODA allows you to specify the four supported verbs (Create, Retrieve, Delete, and Update) and assign as actions of each verb n plus two methods, where n equals the number of methods in the corresponding CORBA interface. The two additional methods are those supported by the connector (LoadFromProxy and WriteToProxy). 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 CORBA connector, see “Verb ASI” on page 40.

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. Click OK.

Specifying the verb ASI

For each verb selected in Step 1 of “Selecting verbs” on page 57, a separate window appears where you specify the method sequence that must be executed for the verb.

Figure 19 on page 59 illustrates this screen for the Retrieve verb of the Hello business object created in Figure 16 on page 56 and Figure 17 on page 57.

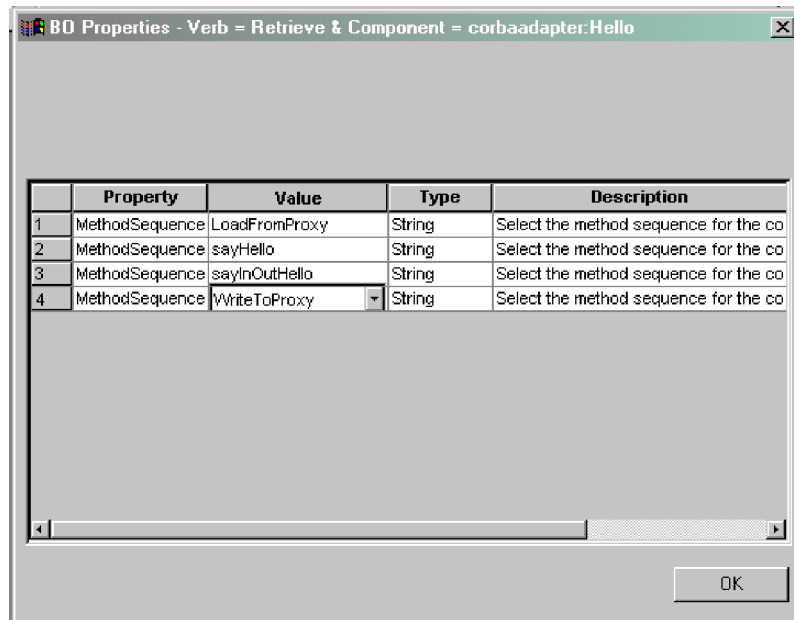


Figure 19. Setting the verb method sequence

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 19, the method sequence is as follows:

- The first method that will be executed in the sequence of methods for the Retrieve verb is LoadFromProxy.
- The second method in the sequence is sayHello.
- The third method in the sequence is sayInOutHello.
- The last method in the sequence is WriteToProxy.

These methods are provided by the CORBA interface (defined in the IDL file), with the exception of two methods that are provided by the ODA: LoadFromProxy and WriteToProxy.

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.

2. Click **OK**.

For a list of the keywords supported by the CORBA verb ASI, see Table 10 on page 41.

Open the business object in a separate window

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

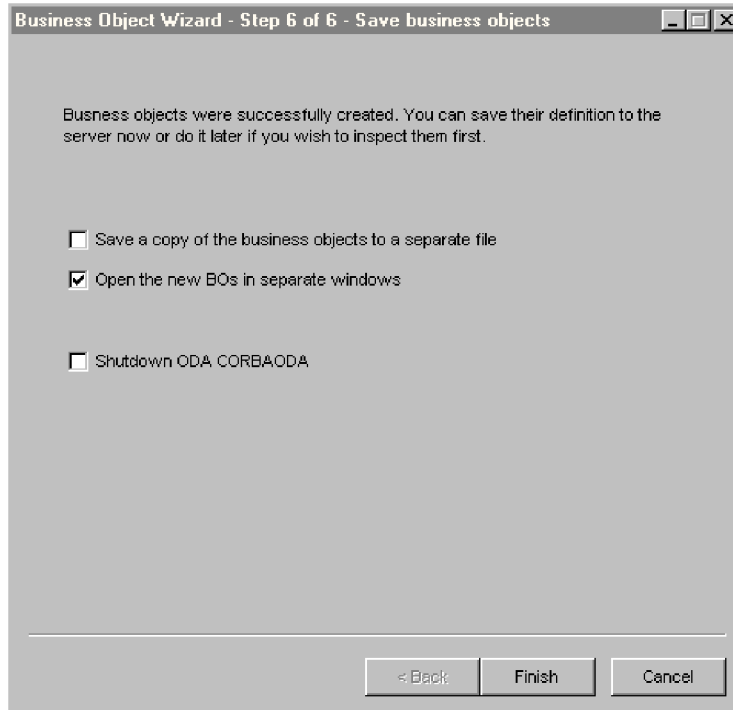


Figure 20. Save business objects screen

You can optionally open the business object in a separate window 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**. A dialog box appears.
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 “Specifying business object information” on page 57.

To save the business objects to a file (only after you have specified a key for the parent-level business object):

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 CORBA ODA” 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 CORBA connector, see “Attribute-level ASI” on page 42.

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 CORBA object attributes are represented as simple attributes and their ASI contains the original CORBA attribute name and type.

For each attribute, the screen provides the name of the attribute, its type, and the ASI information. The `sayHello` attribute of the business object has an ASI that maps the attribute to the original CORBA IDL method name. In this example, the original method name is indicated under the **App Spec Info** column, by the `method_name=sayHello` ASI.

On the **Attributes** tab, you need to specify a key (required by Business Object Designer to validate and save a business object) for each business object for which the ODA has not already specified a key. Note that for certain CORBA types (for example, `CORBA_Short`, `CORBA_Boolean`, and `CORBA_Char`), the ODA sets the keys. Other CORBA types require that you set the key.

You can also use this screen to set child object keys as needed and 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 new business object through the ODA (running in Business Object Designer), do not configure the foreign key in this manner. The foreign key is non-ASI meta data 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 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 39.

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 proxy class that represents this business object. The connector uses this information to map a proxy class to a business object, and, in the case of a server-side business object (when the connector also runs as a server), the connector uses this information to map an implementation class to a business object.

This screen also lists all the verbs that are supported by the business object and provides the ASI for each verb, as it was defined in “Specifying the verb ASI” on page 58.

Figure 21 on page 62 illustrates the business object-level ASI for the Hello business object. The only verb that will execute a method sequence for this business object is Retrieve, which has a verb ASI with the method sequence illustrated here (it was originally set in Figure 19 on page 59).

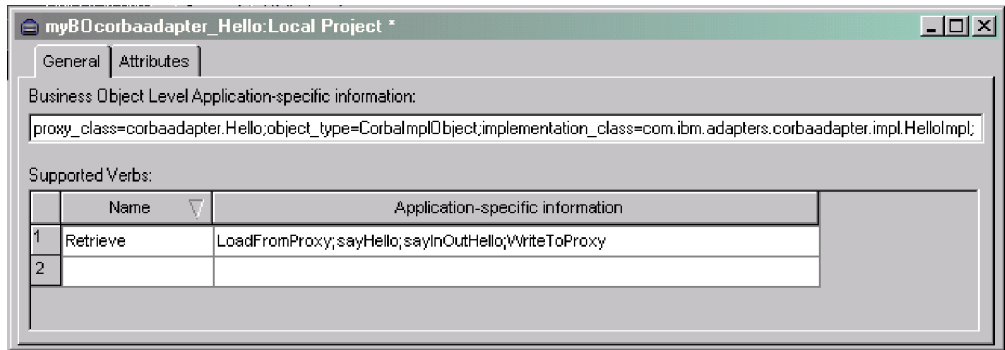


Figure 21. Setting the business object level ASI

On this screen you can modify the ASI of the business object and its supporting verbs.

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 need to upload them to the repository on the server, refer to the *Implementation Guide for WebSphere InterChange Server*.
- **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 *Implementing Adapters with WebSphere MQ Integrator Broker*
- **WebSphere Application Server:** For details, see *Implementing Adapters with WebSphere Application Server*

Chapter 6. Troubleshooting and error handling

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

- “Error handling”
- “Troubleshooting tips” on page 66
- “Logging” on page 66
- “Tracing” on page 66

Error handling

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

```
Message number  
Message text
```

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

Connector error handling

CORBA exception

If the CORBA application is down, or the CORBA call returns a failure, the connector throws a CORBA exception.

The connector handles such exceptions by logging and returning a FAIL code. To aid in debugging, the connector logs the details of the CORBA exception, and returns it in the message field of the `VerbProcessingFailed` exception. The exception also contains information about which call in the sequence failed.

ClassNotFoundException for proxy

When the Loader receives the proxy class name and tries to create a proxy object of that class, an exception is raised if it cannot find the class. The connector logs the error, which includes the name of the class not found, and returns a FAIL code.

InstantiationException in Loader

When the Loader receives the proxy class name and tries to create a proxy 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 by the IDLJ compiler tool.

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 proxy object.

The connector logs the error and returns a FAIL code.

InvocationTargetException in Invoker

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

The connector logs the error and returns a FAIL code.

Invalid argument (CXIgnore) in an InParameter 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. For details about business object structure and verb ASI, see Chapter 4, "Understanding business objects," on page 35.

The connector logs the error and returns a FAIL code.

Cast failure or wrong attribute type

The connector raises an exception if a proxy 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, or a child business object that specifies CBOH (custom BO handler) for an active verb.

The connector logs the error and returns a FAIL code.

ODA error handling

Error produced in buffered reader is: {1}

This exception is raised when an error occurs in the javac compiler. The description of the error appears in {1}.

There are no components or methods selected by the user

The ODA raises this exception if the user has not selected CORBA methods or components in Business Object Designer for creating corresponding business objects.

An error has occurred in {1}:{2}

This exception is raised when an unexpected error occurs in an ODA routine. The module or function that generates the error appears in {1}, and the description of the error appears in {2}.

There are no java files in the specified directory, {1}

The ODA raises this exception when the IDLJ compiler tool does not output any files to the directory for Java proxy files. The name of the directory appears in {1}. This directory is specified in the `OutputFileDir` property of the ODA Configure Agent. For details about this property, see Table 15 on page 54.

The specified directory, {1}, doesn't exist

The ODA raises this exception when the user specifies a value in the `OutputFileDir` property of the ODA Configure Agent but the directory does not exist on the machine where the IDLJ compiler tool outputs the proxy .jar files. The name of the directory appears in {1}. For details about this property, see Table 15 on page 54.

Could not get IDL type information on {1} attribute

The ODA raises this exception when it cannot determine the data type of the CORBA attribute specified in the IDL file. The name of the attribute appears in {1}. Note that the CORBA attribute and business object attribute have the same name (except for the prefix added to the business object name).

{1}: Path to IDL Compiler is incorrect: {2}

The ODA raises this exception when an incorrect pathname has been specified in the `IDLToJavaTool` property of the Configure Agent. For details about this property, see Table 15 on page 54. The module or function that generates the error appears in {1}, and the incorrect pathname appears in {2}.

{1}: Path to IDL Compiler is unspecified

The ODA raises this exception when a pathname is not specified in the `IDLToJavaTool` property of the Configure Agent. For details about this property, see Table 15 on page 54. The module or function that generates the error appears in {1}.

{1}: Could not execute IDLJ. Make sure it is in your PATH

The ODA raises this exception when it cannot run the IDLJ compiler tool. The module or function that generates the error appears in {1}.

Make sure that the directory containing the IDLJ compiler tool is specified in your system path, or, alternatively, make sure you specify the full path of the IDLJ compiler tool in the `IDLToJavaTool` property. For details about this property, see Table 15 on page 54.

Invalid input file found -- <Type>Operations.java is expected

The ODA raises this exception when it attempts to generate a business object that corresponds to an IDL interface but is unable to find a particular Java file.

Unable to create base directory

The ODA raises this exception if it cannot, for whatever reason, create the directory for the implementation classes that are used for server-side business objects (processed with the connector runs as a server).

Directory not found and is not absolute path

The ODA raises this exception if a relative path, rather than an absolute directory path, has been specified in any of the Configure Agent properties. You must specify absolute paths that point to a specific drive and directory.

For details about the Configure Agent properties, see Table 15 on page 54

Unable to create directory {1}

The ODA raises this exception if it cannot, for whatever reason, create a given output directory for any of the business object output files. The directory name appears in {1}.

{1}: Found IDL type of "{2} " in file {3}. This is not a known IDL type. Could not determine IDL type

The ODA raises this exception if an unrecognized type exists in the IDL file. The module or function that generates the error appears in {1}, the name of the unrecognized data type appears in {2}, and the IDL file name appears in {3}.

This exception is typically raised if you are using a Java proxy compiler other than the IDLJ compiler tool of IBM Java ORB, which is the only IDL to Java proxy compiler that the connector currently supports.

Troubleshooting tips

Problem	Potential solution / explanation
The ODA Select Source screen (see Figure 16 on page 56) appears blank with no CORBA objects to select for generating business objects.	Verify which IDL to Java proxy class compiler you are using, and make sure that it is supported by the connector. Currently, the connector supports the IDLJ compiler tool of IBM Java ORB. Verify that the pathname for the IDLJ compiler tool is specified correctly in the IDLToJavaTool property of the Configure Agent. For details about this property, see Table 15 on page 54.
The message "{1}: Found IDL type of "{2} " in file {3}. This is not a known IDL type. Could not determine IDL type" appears because a CORBA fixed data type has been found during IDL compilation.	The IBM IDLJ compiler tool does not support the CORBA fixed data type, hence the ODA does not support generating a business object with an attribute that represents a fixed data type. Verify which IDL compiler you are using, and make sure that it is supported by the connector. Currently, the connector supports the IBM IDLJ compiler tool.
The ODA failed to generate Java proxy files from the CORBA IDL file, for whatever reason.	Run the IDLJ compiler tool (idlj.exe for Windows; idlj for Unix) manually, outside of the ODA, and then compile the resulting Java files manually, using javac.

Logging

All messages described in "Error handling" on page 63 must be read from the message file (BIA_CORBAConnector.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 21. For more information on tracing, including how to enable and set it, see the *Connector Development Guide*.

Table 16 lists the recommended content for connector tracing message levels.

Table 16. 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.

Table 16. Tracing messages content (continued)

Level	Description
Level 1	Use this level for trace messages that: <ul style="list-style-type: none"> • 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.
Level 2	Use this level for trace messages that: <ul style="list-style-type: none"> • 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: <ul style="list-style-type: none"> • 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: <ul style="list-style-type: none"> • 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: <ul style="list-style-type: none"> • 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 sections on Connector Configurator in this guide.

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

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.
- **Agent restart (ICS only)**
The change takes effect only after you stop and restart the application-specific component.
- **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.

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 Table 17 on page 71 below.

Summary of standard properties

Table 17 on page 71 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.

Note: In the "Notes" column in Table 17 on page 71, the phrase "Repository directory is REMOTE" indicates that the broker is the InterChange Server. When the broker is WMQI or WAS, the repository directory is set to LOCAL

Table 17. 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> (broker is ICS)
AgentTraceLevel	0-5	0	Dynamic	
ApplicationName	Application name	Value specified for the connector application name	Component restart	
BrokerType	ICS, WMQI, WAS		Component restart	
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> (broker is ICS)
ContainerManagedEvents	No value or JMS	No value	Component restart	Delivery Transport is JMS
ControllerStoreAndForwardMode	true or false	true	Dynamic	Repository directory is <REMOTE> (broker is ICS)
ControllerTraceLevel	0-5	0	Dynamic	Repository directory is <REMOTE> (broker is ICS)
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

Table 17. Summary of standard configuration properties (continued)

Property name	Possible values	Default value	Update method	Notes
DuplicateEventElimination	true or false	false	Component restart	JMS transport only: Container Managed Events must be <NONE>
FaultQueue		CONNECTORNAME/FAULTQUEUE	Component restart	JMS transport only
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> (broker is ICS)
JvmMaxNativeStackSize	Size of stack in kilobytes	128k	Component restart	Repository directory is <REMOTE> (broker is ICS)
JvmMinHeapSize	Heap size in megabytes	1m	Component restart	Repository directory is <REMOTE> (broker is ICS)
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	

Table 17. Summary of standard configuration properties (continued)

Property name	Possible values	Default value	Update method	Notes
LogAtInterchangeEnd	true or false	false	Component restart	Repository Directory must be <REMOTE> (broker is ICS)
MaxEventCapacity	1-2147483647	2147483647	Dynamic	Repository Directory must be <REMOTE> (broker is ICS)
MessageFileName	Path or filename	CONNECTORNAMEConnector.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> (broker is ICS)
OADMaxNumRetry	A positive number	1000	Dynamic	Repository Directory must be <REMOTE> (broker is ICS)
OADRetryTimeInterval	A positive number in minutes	10	Dynamic	Repository Directory must be <REMOTE> (broker is ICS)
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	

Table 17. Summary of standard configuration properties (continued)

Property name	Possible values	Default value	Update method	Notes
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.
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 (milliseconds)	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>
WsifSynchronousRequestTimeout	0 - any number (milliseconds)	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 (Object Request Broker) connections opened by `orb.init[]`.

The default value of this property is set to 1. You can change it as required.

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-down list. To add other supported values to the drop-down list, you must manually modify the `\Data\Std\stdConnProps.xml` file in the product directory. For more information, see the sections on Connector Configurator in this guide.

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.

There is no default value.

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

- PollQuantity = 1 to 500
- SourceQueue = /SOURCEQUEUE

You must also configure a data handler with the MimeType, DHClass (data handler class), and DataHandlerConfigMOName (the meta-object name, which is optional) properties. To set those values, use the **Data Handler** tab in Connector Configurator.

These properties are adapter-specific, but **example** values are:

- MimeType = text/xml
- DHClass = com.crossworlds.DataHandlers.text.xml
- DataHandlerConfigMOName = MO_DataHandler_Default

The fields for these values in 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 the `RepositoryDirectory` is remote, 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`. Use the default when connecting to a local message broker.

When you connect to a remote message broker, this property takes the following (mandatory) values:

`QueueMgrName:<Channel>:<HostName>:<PortNumber>`,

where the variables are:

`QueueMgrName`: The name of the queue manager.

`Channel`: The channel used by the client.

`HostName`: The name of the machine where the queue manager is to reside.

`PortNumber`: The port number to be used by the queue manager for listening.

For example:

```
jms.MessageBrokerName = WBIMB.Queue.Manager:CHANNEL1:RemoteMachine:1456
```

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:

<i>ll</i>	a two-character language code (usually in lower case)
<i>TT</i>	a two-letter country or territory code (usually in upper case)
<i>codeset</i>	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-down list. To add other supported values to the drop-down list, you must manually modify the `\Data\Std\stdConnProps.xml` file in the product directory. For more information, refer to the sections on Connector Configurator in this guide.

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 RepositoryDirectory 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 in the product directory. 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 true to 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

This is the interval between the end of the last poll and the start of the next poll. PollFrequency specifies the amount of time (in milliseconds) between the end of one polling action, and the start of the next polling action. This is not the interval between polling actions. Rather, the logic is as follows:

- Poll to obtain the number of objects specified by the value of PollQuantity.
- Process these objects. For some adapters, this may be partly done on separate threads, which execute asynchronously to the next polling action.
- Delay for the interval specified by PollFrequency.
- Repeat the cycle.

Set PollFrequency to one of the following values:

- The number of milliseconds between polling actions (an integer).
- 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. Where they exist, these restrictions are documented in the chapter on installing and configuring the adapter.

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.

FIX

An email message is also considered an event. The connector behaves as follows when it is polled for email.

Polled once - connector goes to pick 1. the body of the message as it is also considered an attachment also. Since no DH was specified for this mime type, it will ignore the body. 2. connector process first PO attachment. DH is available for this mime type so it sends the business object to the Visual Test Connector. If the 3. accept in VTC again no BO should come thru Polled second time 1. connector process second PO attachment. DH is available for this mime type so it sends the BO to VTC2. accept in VTC again now the third PO attachment should come through. This is the correct behaviour.

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 configurable 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 `JMSand` 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 76.

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 is CwB0.

WsifSynchronousRequestTimeout

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” on page 87
- “Starting Connector Configurator” on page 88
- “Creating a connector-specific property template” on page 89
- “Creating a new configuration file” on page 91
- “Setting the configuration file properties” on page 94
- “Using Connector Configurator in a globalized environment” on page 100

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 88).

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 89 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 Tools>Connector Configurator**.
- Select **File>New>Connector Configuration**.
- 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 93.)

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:

- 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.
- From Connector Configurator, select **File>Open**. Select the name of the connector configuration file from a project or from the directory in which it is stored.
- 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 connector definition as the template.

- To create a new template, see “Creating a new template” on page 89.
- To use an existing file, simply modify an existing template and save it under the new name. You can find existing templates in your `\WebSphereAdapters\bin\Data\App` directory.

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 in Connector Configurator:

1. Click **File>New>Connector-Specific Property Template**.
2. The **Connector-Specific Property Template** dialog box appears.
 - Enter a name for the new template in the **Name** field below **Input a New Template Name**. You will see this name again when you open the dialog box for creating a new configuration file from a template.
 - 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 appears in the **Template Preview** display.
3. You can use an existing template whose property definitions are similar to those required by your connector as a starting point for your template. If you do not see any template that displays the connector-specific properties used by your connector, you will need to create one.

- If you are planning to modify an existing template, select the name of the template from the list in the **Template Name** table below **Select the Existing Template to Modify: Find Template**.
- This table displays the names of all currently available templates. You can also search for a template.

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**, enter your values.

To create a new property value:

1. Select the property in the **Edit properties** list and right-click on it.
2. From the dialog box, select **Add**.
3. Enter the name of the new property value and click OK. The value appears in the **Value** panel on the right.

The **Value** panel 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 **Dependencies - 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, you must name it and select an integration broker.

- 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.
- In stand-alone mode: from Connector Configurator, select **File>New>Connector Configuration**. In the New Connector window, enter the name of the new connector.

You also need 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 **Integration Broker** field, select `ICS`, `WebSphere Message Brokers` or `WAS connectivity`.
- drop-down the remaining fields in the **New Connector** window, as described later in this chapter.

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 name. You can fill in all the field values to drop-down 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 96..
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 displayed for each property. It indicates whether a component or agent restart is necessary to activate changed values. You cannot configure this setting.

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 Connector-specific Properties 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 70.

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 an ICL (Integration Component Library) 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 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 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. By default, the file is saved to \WebSphereAdapters\bin\Data\App.
- You can also save it to a WebSphere Application Server project if you have set one up.

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

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.

COPYRIGHT LICENSE

This information may contain sample application programs in source language, which illustrates programming techniques on various operating platforms. You may copy, modify, and distribute these sample programs in any form without payment to IBM, for the purposes of developing, using, marketing or distributing application programs conforming to the application programming interface for the operating platform for which the sample programs are written. These examples have not been thoroughly tested under all conditions. IBM, therefore, cannot guarantee or imply reliability, serviceability, or function of these programs.

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.



WebSphere Business Integration Adapter Framework V2.4.0



Printed in USA