

IBM WebSphere Business Integration Adapters



Adapter for i2 User Guide

Version 1.3.x

Note!

Before using this information and the product it supports, read the information in Appendix C, "Notices," on page 77.

25June2004

This edition of this document applies to IBM WebSphere Business Integration Adapter for i2 (5724-G91), version 1.3.x.

To send us your comments about IBM WebSphere Business Integration documentation, e-mail 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 2002, 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
Related documents	v
Typographic conventions	vi
New in this release	vii
New in release 1.3.x	vii
New in release 1.2.x	vii
New in release 1.1.x	vii
Chapter 1. Overview of the connector	1
Connector architecture	1
How the connector works	3
Chapter 2. Installing and configuring the connector	7
Adapter environment	7
Installing the adapter and related files	8
Installed file structure	8
Configuring the connector	9
Creating multiple connector instances	11
Starting the connector	13
Stopping the connector	14
Chapter 3. Understanding business objects for the connector	15
Defining connector metadata	15
Overview of business object structure	15
i2 business object structure	16
Specifying business object attribute properties	22
Identifying business object application-specific information	23
Chapter 4. Generating business objects using the i2 ODA	25
Overview of the i2 ODA	25
Installing the i2 ODA	25
Using the i2 ODA in Business Object Designer	27
Create the metaobject for polling	35
Chapter 5. Troubleshooting and error handling	37
Logging error messages	37
Tracing messages	40
Running the adapter against a CIS on a different subnet	41
Tips for troubleshooting	42
Appendix A. Standard configuration properties for connectors	43
New and deleted properties	43
Configuring standard connector properties	43
Summary of standard properties	44
Standard configuration properties	49
Appendix B. Connector Configurator	61
Overview of Connector Configurator	61
Starting Connector Configurator	62
Running Configurator from System Manager	62
Creating a connector-specific property template	63

Creating a new configuration file	65
Using an existing file	66
Completing a configuration file.	67
Setting the configuration file properties	68
Saving your configuration file	73
Changing a configuration file	74
Completing the configuration	74
Using Connector Configurator in a globalized environment	74
Appendix C. Notices	77
Programming interface information	78
Trademarks and service marks	79

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 i2.

Audience

This document is for consultants, developers, and system administrators who use the connector at customer sites.

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 WebSphere InterChange Server as your integration broker:

<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, WebSphere MQ Integrator Broker, 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.

Typographic conventions

This document uses the following conventions:

<i>ProductDir</i>	Represents the directory where the product is installed.
courier font	Indicates a literal value, such as a command name, file name, information that you type, or information that the system prints on the screen.
<i>italic, italic</i>	Indicates a new term the first time that it appears, 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, a pipe separates 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.
< >	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 path names are relative to the directory where the connector for i2 is installed on your system.
<i>%text%</i> and <i>\$text</i>	Text within percent (%) signs indicates the value of the Windows text system variable or user variable. The equivalent notation in a UNIX environment is <i>\$text</i> , indicating the value of the <i>text</i> UNIX environment variable.

New in this release

This section describes the new and changed features of IBM WebSphere Business Integration Adapter for i2 that are covered in this document.

New in release 1.3.x

- Support is no longer provided for Solaris 7.0.
- Connector definition file format has changed from .txt to .xsd.
- Improved handling of unregistering shared connections in event notification.
- Improved connection performance when multiple service requests are made using the same connection ID.

New in release 1.2.x

- Adapter installation information has been moved from this guide. See Chapter 2 for the new location of that information.
- The i2 Adapter is globalized and has DBCS support.
- Support is provided for HP-UX.
- Support is provided for creating multiple connector instances.
- Three new configuration properties have been added to the i2 ODA: MetadataService, PortTypesOperation, and SchemasOperation.
- Support is provided for running the i2 Adapter against a CIS on a different subnet.

New in release 1.1.x

- The CrossWorlds name is no longer used to describe an entire system or to modify the names of components or tools. For example, "CrossWorlds System Manager" is now "System Manager," and "CrossWorlds Interchange Server" is now "WebSphere Interchange Server." Name changes have been implemented, accordingly, in this document.
- Support is provided for using WebSphere Application Server (WAS) as an integration broker.
- Support is provided for authentication, both at the global level using the Application Username and Password in the connector configuration properties, and at the business object level using the MO_instance object attributes username and password.
- Support is provided for stateful operations, using the stateful operation request business object, to allow the CIS agent to maintain session state.
- Support is provided for guaranteed messaging and delivery and persistent client connections.
- Support is provided for interaction resolution for operations which failed to execute within the specified execution time.
- Sample polling business objects have been enhanced.
- Enhancement updates are provided for the i2 ODA, as follows:
 - Removal of the VisiBroker OAD from the launching information
 - Distinction between 'Types' and 'Formats'
 - Generation of a business object, despite existence of an "any" tag

– Screen shots for the ODA wizard

Chapter 1. Overview of the connector

This chapter describes the connector component of the IBM WebSphere Business Integration Adapter for i2 and the relevant business integration system architecture.

The i2 connector integrates with i2 application modules through i2's Common Integration Services (CIS) API. CIS API from i2 is an implementation of JCA Common Client Interface. i2 has a suite of application modules that support CIS. The i2 connector is metadata driven, has object discovery capability, and enables integration to any version 6.0.1 SDK CIS-enabled i2 application. Many i2 modules versions 5.2 and above, support version 6.0.1 CIS SDK. This connector is available on Windows, Solaris, AIX and HP-UX.

Connectors consist of two parts: the *application-specific component* and the *connector framework*. The application-specific component contains code tailored to a particular application or technology (in this case, i2). The connector framework, whose code is common to all connectors, acts as an intermediary between the integration broker--WebSphere InterChange Server (ICS), WebSphere MQ Integrator Broker (WMQI), or WebSphere Application Server (WAS)--and the application-specific component. 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

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

For more information about the relationship of the integration broker to the connector, see the *System Administration Guide* (for ICS), *Implementing Adapters with WebSphere MQ Integrator Broker*, or *Implementing Adapters with WebSphere Application Server*.

This chapter contains the following sections:

- "Connector architecture" on page 1
- "How the connector works" on page 3

Connector architecture

i2's Common Integration Service (CIS) enables connectivity between external applications and i2 application modules.

CIS includes three primary components:

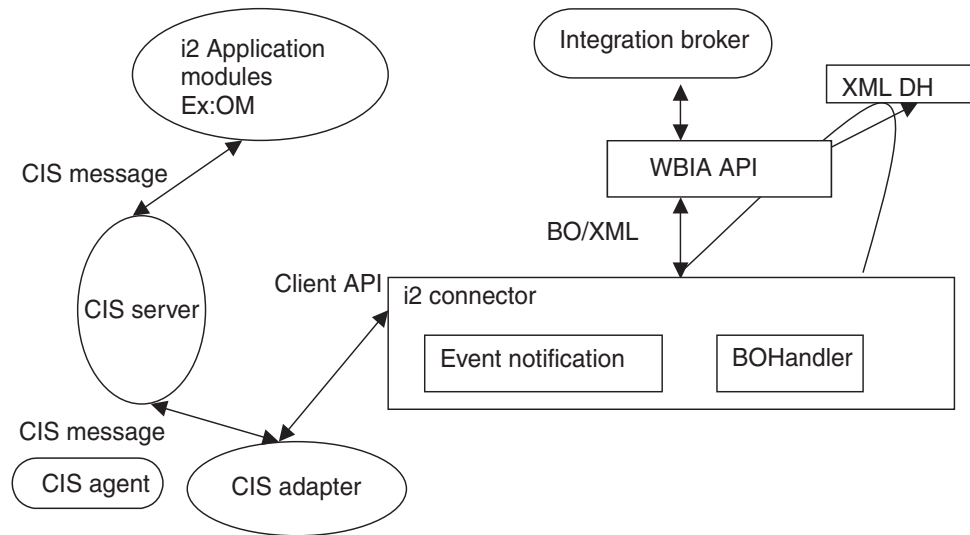
- CIS Front Bus--used by applications to specify an XML metadata format interface that details the available functions and their expected input and output data. CIS scripts generate the required representations of input and output data in an XML schema and Java Beans. Product teams implement these functions by writing handlers in Java that implement standard CIS interfaces and perform the required logic. These handlers can deal with data as XML or Java Beans. CIS

infrastructure deploys these interfaces so that the client can invoke this functionality from a variety of resources.

- CIS Back Bus--used by applications to create a bulk import/export interface for data transfers.
- CIS Single Sign-On--a standard set of Java interfaces used by Web applications to authenticate users against a central authentication. store.

The i2 connector interacts with the CIS Front Bus using the CIS Client API provided by i2 along with its CIS adapters. The CIS Client API is the implementation of JCA Common Client Interface. CIS adapters operate based on CIS metadata information using the various bindings.

The following diagram shows the i2 connector and components of the i2 framework.



The following table describes the terminology used for the components of the i2 connector and framework.

Component	Description
CIS	Common Integration Services provided by i2 to enable connectivity between the external applications and i2 application modules
OM	Order Management, just an example of an i2 application module
CIS agent	CIS server application that runs on a central server. It maintains connection information about client applications and manages and monitors all the client applications that are part of the solution. The CIS adapter and i2 application modules register with the CIS agent.
JCA CCI	Java Connector Architecture's Common Client Interface
CIS adapter	i2's CIS Client API and implementation of JCA-CCI
WBIA API	API used by the i2 connector to communicate with the designated integration broker.

Component	Description
integration broker	A program that handles the execution of the business object processing logic. Supported brokers are InterChange Server (ICS), WebSphere MQ Integrator Broker (WMQI), and WebSphere Application Server (WAS).
XML DH	IBM's data handler (DH) used to transform XML messages to the IBM business objects and vice versa. You need to configure the XML DH for use with the i2 connector.
CIS server	Integration container which handles operation invocations. Integration container and CIS server are used interchangeably in this document.

How the connector works

The i2 connector is a CIS (Common Integration Services) client. It connects to the CIS client API in a non-managed environment, that is, it connects to the CIS adapter directly without an application server using the application's user name and password for authentication. For added flexibility, authentication is also possible during request service processing using the credential information included in the business object. For more information, see Chapter 3, "Understanding business objects for the connector," on page 15.

At present, the CIS client API does not support any connection pooling mechanism. There is no limitation on the number of connections. The i2 CIS adapter provides guaranteed messaging and delivery and persistent client connections, depending on the configuration settings of the i2 CIS agent. Data which includes double byte characters will remain consistent when used with the i2 connector. It supports the Group 1 Languages.

The i2 connector is bi-directional. It can process events originating from i2 applications, as well as requests sent by the broker to the application.

For event subscriptions, the i2 connector uses the information in the i2 metaobjects. It registers the operations on the types specified in these metaobjects with i2's CIS agent through the CIS adapter. The CIS agent listens to the registered operations and detects event messages only for registered operations. The i2 connector obtains these messages with the poll call.

For request processing, the i2 connector processes the requests coming from the integration broker by transforming incoming business objects into CIS records and using the appropriate CIS Client API calls to execute the operation on the i2 application modules. Request processing for the i2 connector is multi-threaded, it can handle multiple requests.

The i2 connector follows the metadata design principles outlined in the *IBM Connector Developer's Guidelines*. This means new IBM business objects can be defined without additional coding or customization at the i2 connector code level. For more information, see Chapter 3, "Understanding business objects for the connector," on page 15.

Processing subscriptions

The following sections describe how the connector processes application events.

Event detection and notification

Events for the purpose of this document are the CIS messages published from the i2 application modules. The i2 application notifies the connector of all the events occurring in the modules for which corresponding operations have been registered with the CIS agent.

The onus of registering the operations of interest lies with the i2 connector.

Example: If the i2 connector is interested in the Bidding type operation addBid, any new Bidding additions to the i2 application module will be queued in the CIS server once the i2 connector registers for the addBid operation.

The i2 connector uses the information in the i2 metaobjects. It registers its intent to listen to some of the operations with the poll call. Effectively, this tells the CIS agent that the i2 connector wants to check for the output of the registered operations.

Status updates

No status updates are made to the i2 applications. Typically, the event status, for example, SUCCESS, FAIL, UNSUBSCRIBED, is written to the application's event store. Since no event store is maintained for i2, the status update strategy is not relevant for the i2 connector. Error messages, if any, are logged to the i2 adapter log file. For more information, see Chapter 5, "Troubleshooting and error handling," on page 37.

Event retrieval

For the i2 connector, polling is single threaded. The connector uses i2 metaobjects to register the operations of interest with the CIS agent for polling. These metaobject names have the i2MO prefix and store information about the operation and the corresponding IBM wrapper business object name for the specified operation and type. The attributes for the metaobject are specified as static default values. Default value is an attribute property, which can be set at the business object design time. For information on the wrapper business object structure and attribute properties, see Chapter 3, "Understanding business objects for the connector," on page 15 and Chapter 4, "Generating business objects using the i2 ODA," on page 25.

The steps involved in retrieving a subscription message are as follows:

1. The i2 connector uses a connection that is authenticated using the Application Username and Password connector properties, or the username and password found in the polling metaobject.
2. The i2 connector registers the operations with the CIS agent after reading the i2MO metaobject information. The information in the metaobjects is cached by the i2 connector with the first poll call. The connector allocates a connection for this metaobject using the username, password, instanceId and connectionId provided in the metaobject.
3. Each poll call is issued from the integration broker, based on the connector property PollFrequency. In case there were any registration failures in the first poll call, the i2 connector tries to register the same operation with the subsequent poll calls.

Once the i2 connector registers its interest for data using a persistent connection, the i2 connector can reliably retrieve data that was sent out when it was not running. The EventSubscriptionConnectionID property value or the polling metaobject connectionId attribute enables it to keep its association with the subscription through restarts of the connector.

For information on configuring the CIS agent with persistent storage, see your i2 CIS infrastructure guide. For information on configuring the EventSubscriptionConnectionID property, see Chapter 2, “Installing and configuring the connector,” on page 7.

4. With all the poll calls, the i2 connector checks on the output of the operations that it has registered with the CIS agent. If there is any output from any of the operations, it retrieves the output in the form of a CIS record. The i2 connector retrieves the PollQuantity (connector property) number of messages for each poll call for each registered operation.

Example: If the PollQuantity is set to 5 and there are 5 registered operations, each poll call will result in checking the output 25 times. If the PollQuantity is not set, a default of 1 message is retrieved for each poll call for each operation.

5. The retrieved XML message is converted to a business object. The business object is set as the child attribute in the wrapper business object for the operation. The instanceId from which this output was retrieved is set as the instanceId in the metaobject attribute of the wrapper. For more information, see Chapter 3, “Understanding business objects for the connector,” on page 15.
6. The connector sends the wrapper business object to the integration broker for further processing.

Processing verbs (operations)

Operations are i2’s equivalent for verbs and are defined by the XML structure provided by i2 for each port. For more information, see Chapter 3, “Understanding business objects for the connector,” on page 15, and Chapter 4, “Generating business objects using the i2 ODA,” on page 25.

Processing service call requests

When the i2 connector receives a service call request from an integration broker to perform an operation in an application, the request takes the form of a wrapper business object. The wrapper business object encompasses a metaobject, an operation or verb, and input and/or output types or formats for the operation. The metaobject (MO_Instance) and the input and output business objects are the child business objects of the wrapper business object. Enclosed in the MO_Instance object are the attribute values for the instanceId, connectionId, username, and password. The username and password are used for authentication during request processing. The verb for the wrapper business object must be a valid operation for the specified instance.

The information about the child business object, whether it is an input or output type, is obtained from the Application Specific Information (ASI) of the wrapper business object’s attributes.

Example: ASI Type=input indicates that the child business object is of input type.

The input child business object is first converted to an XML message by the XML data handler. The business object is then transformed into a CIS record using the CIS utility. Then the operation is executed using the CIS Client API.

If the operation sends some output XML message, it is converted to an output child business object; and the output child business object in the wrapper business object is populated with the appropriate value.

If the i2 CIS agent is running with persistence enabled, the CIS agent can use stateful operations to maintain the session state. The connector can retrieve results

of operations that failed to execute within the specified execution time and attempt to resolve the operations. It returns the results to the integration broker.

You can control the number of times that an interaction should be attempted to be resolved by setting the value for the `InteractionResolutionAttempts` property. A value of zero (0) for the `InteractionResolutionAttempts` prevents the connector from trying to resolve an interaction. Interaction Resolution attempts are only made on interactions with the CIS adapter state of `INDOUBT`.

For information on stateful operations, see “Using metaobjects for stateful operations” on page 18. For information on configuring the `InteractionResolutionAttempts` and `ExecutionTimeout` properties, see Chapter 2, “Installing and configuring the connector,” on page 7.

Status updates

Any error conditions that occur while processing are logged as detailed error messages in the adapter log.

ReturnStatusDescriptor: The connector populates a structure called the `ReturnStatusDescriptor` with the message and status of the latest error that occurred during the processing of a service call request. You can access the `ReturnStatusDescriptor` to find the reason for the cause of a failure during a service call request. The adapter framework propagates the structure, as appropriate.

When using persistent connections with the CIS adapter, status values have the range of `CREATED`, `STARTED`, `UNDELIVERABLE`, `DELIVERED`, `SUCCESS`, `INDOUBT`, `FAILED` to indicate the state of the interaction on a request processing failure.

Chapter 2. Installing and configuring the connector

This chapter describes how to install and configure the connector component of IBM WebSphere Business Integration Adapter for i2 and how to configure applications to work with the connector. It contains the following sections:

- “Adapter environment” on page 7
- “Installing the adapter and related files” on page 8
- “Configuring the connector” on page 9
- “Creating multiple connector instances” on page 11
- “Starting the connector” on page 13
- “Stopping the connector” on page 14

Adapter environment

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

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 1.3.x of the Adapter for i2 is supported on the following adapter framework and integration brokers:

- **Adapter framework:** WebSphere Business Integration Adapter Framework, versions 2.1, 2.2, 2.3.x, and 2.4
- **Integration brokers:**
 - WebSphere InterChange Server, versions 4.1.1 and 4.2.x
 - WebSphere MQ Integrator, version 2.1.0
 - 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.

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, WebSphere MQ Integrator Broker, 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>

Required hardware and software

The adapter requires the following hardware and software:

- Applications:
 - CIS SDK 6.0.1
 - J2EE.jar
 - Appropriate CIS adapter (MetadataService adapter is required for the i2 ODA)
- One of the following application platforms:
 - Windows: 2000

Note: Beginning with version 1.2.x, the Adapter for i2 is no longer supported on Microsoft Windows NT.

- UNIX: Solaris, AIX, or HP
- One of the following adapter platforms:
 - Windows: 2000
 - UNIX: Solaris 8.0; AIX 5.1 or 5.2; or HP-UX 11i

Processing locale-dependent data

The connector has been internationalized so that it can support delivery of double-byte character sets (DBCS) going into an interface that also supports double-byte character sets, and deliver message text in the specified language. When the connector transfers data from a location that uses one character code to a location that uses a different code set, it performs character conversion to preserve the meaning of the data.

The Java run time environment within the Java Virtual Machine (JVM) represents data in the Unicode character code set. Unicode contains encodings for characters in most known character code sets (both single-byte and multibyte). Most components in the WebSphere business integration system are written in Java; therefore, when data is transferred between most integration components, there is no need for character conversion.

Installing the adapter 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>

Installed file structure

The following table describes the file structure used by the connector and identifies the installed files.

Notes:

1. This document uses (\) backslashes as the convention for directory paths. For UNIX installations, substitute slashes (/) for backslashes.
2. All product path names are relative to the directory where the product is installed on your system.

- For Windows, Installer adds an icon for the connector file to the IBM WebSphere Business Integration Adapters menu. For a fast way to start the connector, create a shortcut to this file on the desktop.

Subdirectory of <i>ProductDir</i>	Description
connectors\i2	Contains the connector CWi2.jar file, which has the connector code and the start_i2.bat files (WIN) or start_i2.sh files (UNIX).
connectors\messages	Contains the i2Adapter.txt file for error messages.
repository\i2	Contains the CN_i2.txt file.
connectors\i2\Sample	Contains sample files for creating business objects.

For more information on installing the connector component, see one of the following guides, depending on the integration broker you are using:

- *System Installation Guide* for your platform (when using InterChange Server as the integration broker)
- *Implementing Adapters with WebSphere MQ Integrator Broker* (when using WebSphere MQ Integrator as the integration broker)
- *Implementing Adapters with WebSphere Application Server* (when using WebSphere Application Server as the integration broker)

Configuring the connector

Connectors have two types of configuration properties: standard configuration properties and connector-specific configuration properties. You must set the values of these properties before running the connector. As you enter the configuration values, they are saved in the repository.

To configure connector properties, use one of the following tools:

- Connector Designer--if InterChange Server is the integration broker

Tip: Access this tool from System Manager.

- Connector Configurator--if WebSphere MQ Integrator is the integration broker

Tip: Access this tool from the IBM WebSphere Business Integration Adapter program folder.

For more information about Connector Configurator, see Appendix B, "Connector Configurator," on page 61.

A connector obtains its configuration values at startup. During a run-time session, you may want to change the values of one or more connector properties. Changes to some connector configuration properties, such as AgentTraceLevel, are dynamic, taking effect immediately. Changes to other connector properties are static, requiring component restart or system restart after a change. To determine whether a property is dynamic or static, refer to the update method column in Connector Designer.

Standard connector properties

Standard configuration properties provide information that all connectors use. For detailed information about these properties, see Appendix A, "Standard configuration properties for connectors," on page 43.

Note: Because the connector for i2 supports the InterChange Server, WebSphere MQ Integrator, and WebSphere Application Server integration brokers, configuration properties for all three brokers are relevant to the connector.

In addition, the following supplemental information on standard connector properties applies to i2.

LogAtInterchangeEnd

Tells whether to log errors on the InterChange Server (ICS).

The default value is false.

MessageFileName

Path of the error message file if it is not located in the standard message location *ProductDir\connectors\messages*. If the message file name is not in a fully qualified path, the message file is assumed to be located in the directory specified by the HOME environment variable or the startup parameter *user.home*. If a connector message file does not exist, the WBIA API message file is used. If that file does not exist, the *InterchangeSystem.txt* file is used as the message file.

The default value is *i2Adapter.txt*.

Connector-specific properties

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

The following table lists the connector-specific configuration properties for the connector along with their descriptions and possible values.

Property	Description	Possible values
ApplicationName	<i>Unique name specified for each connector</i>	i2 adapter
ApplicationUserName	<i>User name for the i2 connection used for authentication</i>	i2User
ApplicationPassword	<i>Password for the i2 connection used for authentication</i>	i2Password
CISAgentHostName	<i>Name used when the CIS agent is running on a remote machine. If it is not set, the current local host is assumed to have the CIS agent running. If it is set, the i2 connector establishes a connection with this remote host.</i>	String host name Example: any machine name like California
ExecutionTimeout	<i>Time in milliseconds before the call to i2 application terminates.</i>	Default is 30000
EventSubscriptionConnectionId	<i>Default connectionId to be used for request processing</i>	EventConn1
InteractionResolutionAttempts	<i>Number of times that an interaction should be attempted to be resolved</i>	Default is 2
PollQuantity	<i>Number of messages that will be retrieved from the client queue while polling for each registered operation; it will be pollQuantity multiplied by the number of registered operations.</i>	Default is 1

Property	Description	Possible values
UseDefaults	Value that the connector checks for to identify the default value of the attributes during request processing. This is not used by the i2 connector.	Not required for this connector

Configuring start_i2.bat (for Windows) or start_i2.sh (for UNIX)

You need to add the proper path to the start files for CIS-SDK and j2ee.jar.

Example: The following path information needs to be added to the start_i2.bat file. These are just examples. You should change the path information depending on your local installation.

```
set I2_CIS_HOME_DIR=C:\i2\CIS\6.0.1\cis-sdk
set J2EE_PATH=C:\J2EE_JAR
set i2PROPERTIES="%i2_CIS_HOME_DIR%\properties;
```

(The last line refers to the contents of the "properties" directory of your i2 CIS.)

Note: If your CIS agent is configured for guaranteed messaging and an adapter uses MQ Bindings, you will need to have the .bindings file in the same relative path as your i2 CIS install. This is the jndiProviderURL attribute of your MQ Binding configuration element.

Configuring the data handler

You also need to configure the data handler. Set the following values for the child business object text/xml in MO_DataHandler_Default:

Validation	false
ClassName	com.crossworlds.DataHandlers.text.xml
UseNewLine	false
InitialBufferSize	any appropriate value like 2097152
DummyKey	1

Note: The rest of the fields should be blank.

For detailed information about data handler configuration, see the *Data Handler Guide*.

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 11.
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 1 shows.

Table 1. Startup scripts for a connector

Operating system	Startup script
UNIX-based systems	connector_manager_ <i>connName</i>
Windows	start_ <i>connName</i> .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.

Chapter 3. Understanding business objects for the connector

This chapter describes the structure of i2 business objects, how the connector processes the business objects, and the assumptions the connector makes about them. Use this information as a guide to modifying existing business objects for i2 or as suggestions for implementing new business objects.

The chapter contains the following sections:

- “Defining connector metadata” on page 15
- “Overview of business object structure” on page 15
- “i2 business object structure” on page 16
- “Specifying business object attribute properties” on page 22
- “Identifying business object application-specific information” on page 23

For information on the Object Discovery Agent (ODA) utility that automates the creation of business objects for the IBM WebSphere Business Integration Adapter for i2, see Chapter 4, “Generating business objects using the i2 ODA,” on page 25.

Defining connector metadata

The i2 connector is metadata-driven. In the WebSphere business integration system, metadata is application-specific information that is stored in a business object and that helps the connector interact with the application. A metadata-driven connector handles each business object that it supports based on the metadata encoded in the business object definition rather than on instructions hardcoded in the connector. Business object metadata includes the structure of the business object, the settings of its attribute properties, and the content of its application-specific information. Because the connector is metadata-driven, it can handle new or modified business objects without requiring modifications to the connector code.

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 application-specific information.

Therefore, when you create or modify a business object, your modifications must conform to the rules the connector is designed to follow, or the connector will not be able to process new or modified business objects correctly.

Overview of business object structure

In the WebSphere business integration system, a business object definition consists of a type name, supported verbs, and attributes. An application business object is an instance of a business object definition. It reflects a specific application’s data structure and attribute properties.

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

WebSphere Business Integration Adapter 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 the 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 type of the attribute 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 type of the attribute is the same as the type of the array of child business objects.

A hierarchical business object can have simple attributes and can also have attributes that represent a single-cardinality child business object or an array of child business objects. In turn, each of these business objects can contain single-cardinality child business objects and arrays of business objects, and so on.

In each type of cardinality, the relationship between the parent and child business objects is described by the application-specific text of the key attribute of the child object.

i2 business object structure

The i2 IBM business object is the IBM representation of the i2 message. Each type of message has a corresponding IBM business object.

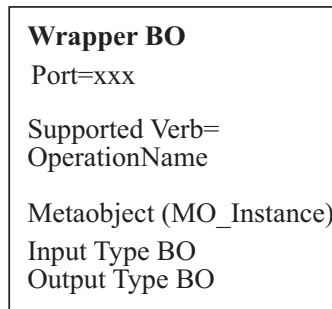
The business objects are generated using the WebSphere Business Integration Adapter utility XML ODA, which reads the XML schema files for these types and generates the corresponding IBM business object. (See Chapter 4, "Generating business objects using the i2 ODA," on page 25 and Chapter 3, "XML data handler" in *IBM WebSphere Business Integration Adapters Data Handler Guide*.)

The i2 business object is a wrapper business object that encapsulates a metaobject, an operation, and input and/or output data (one or the other, both, or none) types or formats for the operation. The Top Level Object Wrapper includes in the metaobject (MO_Instance) the attribute values for the instanceId, connectionId, username, and password. The username and password are provided for authenticating with a different user during request processing.

Note: This business object credential information is in addition to the Application Username and Password connector configuration properties provided for authentication at the global level. If the values at the metaobject (MO_Instance) level are left blank, the adapter will use the global credentials found at the connector configurator level.

There is one wrapper business object for each operation. For more information, see Chapter 4, "Generating business objects using the i2 ODA," on page 25.

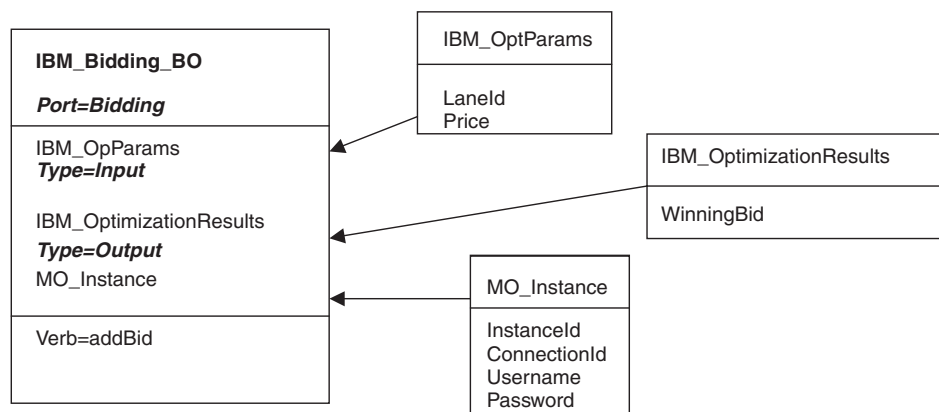
The following diagram shows the parts of a wrapper business object. A description of each part follows the diagram.



- The *metaobject* embedded within the wrapper is used to configure the MO_Instance, which contains values for the attributes *instanceId*, *connectionId*, *username*, and *password*. For more information, see “Configuring metaobjects for polling” on page 18.
- The *operation* is set as the verb on the wrapper business object and is associated with a port. i2 does not have standard verbs. Operations take input and output. If multiple operations have the same set of input and output types, but are supported on different ports, there will be two different wrapper business objects for the different ports.
- The *port* is the name of the i2 port type for the operation.
- The *types*, or alternatively, *formats* are business object attributes which represent data types or formats for an operation.

The following diagram illustrates a sample business object IBM_Bidding_BO, which has three child business objects. In the diagram:

- IBM_OptParams and IBM_OptimizationResults represent the top level business object generated by the XML ODA.
- The application-specific information for the business object is in the *Port* (Bidding) and *Types* (input--IBM_OptParams and output--IBM_OptimizationResults) attributes.
- The operation is addBid.
- The child business objects are:
 - IBM_OptParams, which has two attributes--LaneId and Price
 - IBM_OptimizationResults, which has one attribute--WinningBid
 - MO_Instance, which has four attributes--InstanceId, ConnectionId, Username, and Password



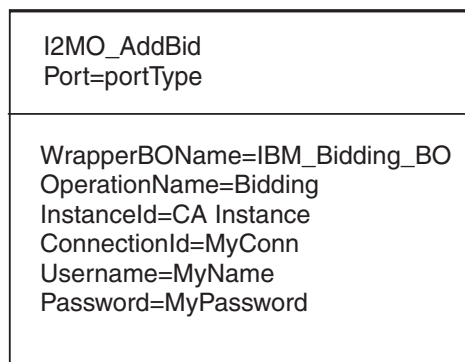
Configuring metaobjects for polling

The connector uses i2 metaobjects to register its interest in specific operations with the CIS agent so that polling can take place. You need to configure one metaobject for each operation of interest.

The metaobject name always starts with i2MO. Each metaobject holds information about the instance that supports the operation and the wrapper business object name for the operation. You need to add a dummy verb to all the metaobjects.

The attributes (instanceId, connectionId, username, password, wrapper business object name, and operation name) within the metaobjects have a static default value. For registering the same operation on a different instance, you either have to change the default value and restart the i2 instance or configure another metaobject for the new instance.

In the following diagram, the metaobject named i2MO_AddBid is used to configure the instanceId CA_Instance, for the Bidding operation addBid, which is set on the wrapper business object named IBM_Bidding_BO. The values shown are default values for the attributes (connectionId, username, and password).



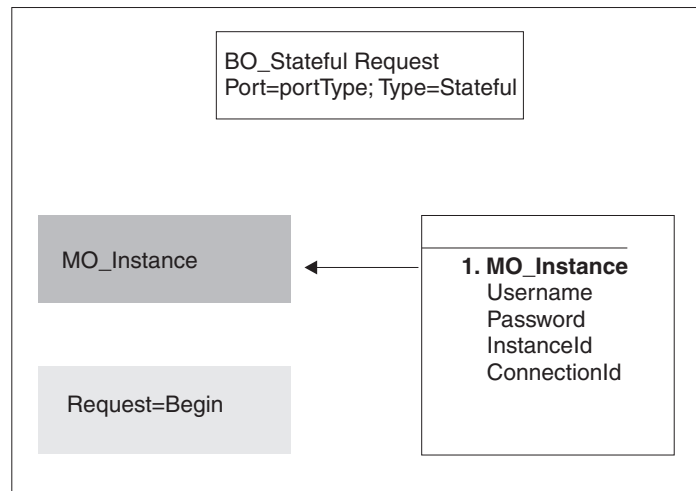
Using metaobjects for stateful operations

Stateful operations allow the i2 connector to maintain the session state throughout a series of operations. These grouped operations are treated as multiple service calls or requests from the perspective of the i2 connector.

To use a stateful operation with the i2 connector, the integration broker requests a stateful connection using the metaobject BO_StatefulRequest that indicates the start and end of a series of stateful operations. The connection is maintained for the stateful operations by associating the connection with a connectionId. This connection is managed for all requests made with the aforementioned connectionId.

Note: While a series of stateful operations are being processed, if the i2 connector crashes the state of these operations is not maintained.

The following diagram shows the metaobject BO_StatefulRequest.



The valid values for the *Request* attribute are "Begin" and "End," signaling to the i2 connector to begin a series of stateful operations with the credential information provided in the MO_Instance. The connectionId can be provided or be generated by the i2 CIS agent. Using the connectionId that is returned from this request, the business processes making subsequent calls to the i2 connector can make stateful operation calls. The application-specific information "Type=Stateful" allows the i2 connector to determine that this business object is a request to start or end the series of stateful operations associated with the enclosed connectionId.

The following examples illustrate using metaobjects for stateful operations.

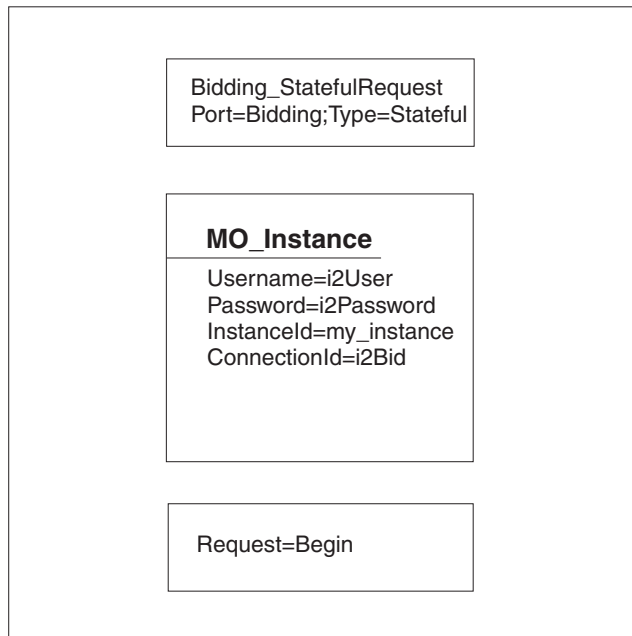
Example 1 of stateful operations

An auction application has a bidding system that uses an i2 CIS adapter to perform some of its auction services. The i2 CIS adapter has stateful operations to keep track of each bid by different users and also return the current winning bid at any time during the bidding process.

To use the i2 connector to make stateful calls and return the winning bid, the integration broker needs a business object to signal the beginning and end of the series of stateful operations (bids). It also needs a business object to submit a bid and another to return the winning bid.

Within the broker the integrator can create a business object based on the metaobject "BO_StatefulRequest", called Bidding_StatefulRequest. The following

diagram shows this metaobject.



Here is an example of how these requests could be made using a broker's business process flow:

A bid has a "LaneId" and a "BiddingAmount", represented by the business object BO_Bid. A call is made to request the current winning bid using the "BO_optimizeBids" business object, which has the "LaneId" as an input business object and the winning bid as the output or returned business object. The returned business object has the "LaneId" and "WinningAmount".

Note: Lines beginning with -->Request processing, indicate how to make the request.

1. Start Bidding--Based on the LaneId, start a series of stateful operations:
-->Request processing, using Bidding_StatefulRequest with "Begin" and connectionId i2Bid.
2. Send Bid1--Send a bid for the LaneId with the BiddingAmount (8):
-->Request processing, using BO_Bid with BiddingAmount.
3. Send Bid2--Send a bid for the LaneId with the BiddingAmount (35):
-->Request processing using BO_Bid with BiddingAmount.
4. Send Bid3--Send a bid for the LaneId with the BiddingAmount (20):
-->Request processing using BO_Bid with BiddingAmount.
5. Send OptimizeBids--Send a request to get the current winning bid:
-->Request processing using BO_OptimizeBids.
Result: The request is processed and the winning bid business object is returned (35).
6. Send Bid4--Send a bid for the LaneId with the BiddingAmount (40):
-->Request processing using BO_Bid with BiddingAmount.
7. Send OptimizeBids--Send a request to get the current winning bid:
-->Request processing using BO_OptimizeBids.

Result: The request is processed and the winning bid business object is returned (40).

8. End Bidding--Send Bidding_StatefulRequest with connectionId "i2Bid" requesting the end of this series of stateful operations:
-->Request processing on Bidding_StatefulRequest with "Request=End".

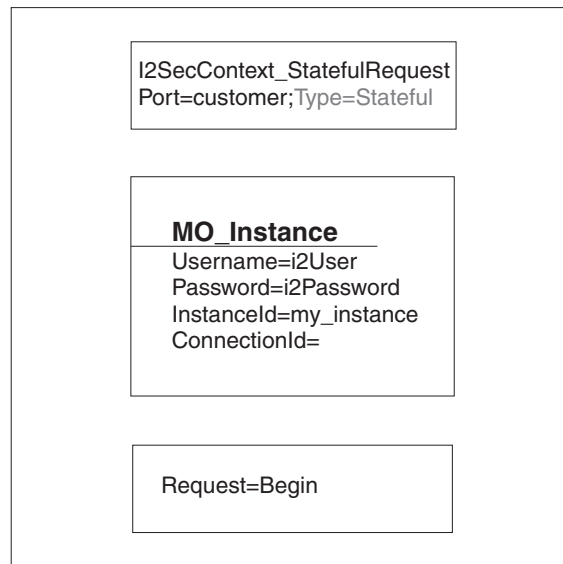
Note: The connectionId is what associates a series of stateful operations with each other; therefore, for each series of bids a different connectionId should be provided. The broker can also allow the i2 CIS agent to generate the connectionId, which will be returned on the "Request=Begin" request call on the i2 connector. Using the returned connectionId, the broker can request a series of stateful operations.

Example 2 of stateful operations

An application requires a special security context to be used by calling clients of the customer configuration processes. An i2 CIS adapter provides a stateful operation that creates and stores the security context for later use by a subsequent call.

The integrator can use the i2 connector to help process these stateful operations using the security context created and provided by the i2 CIS adapter.

A business object called "i2SecContext_StatefulRequest" is created based on the metaobject "BO_StatefulRequest". The following diagram shows the i2SecContext_StatefulRequest metaobject.



Here is an example of how these requests could be made using a broker's business process flow:

Note: Lines beginning with -->Request processing, indicate how to make the request.

1. Start Customer Add--Start a series of stateful operations using i2SecContext_StatefulRequest:
-->Request processing using i2SecContext_StatefulRequest with "Begin".

Result: Since the connectionId was not set in this business object, after the request call, the returned object will contain the connectionId for which these stateful operations need to be associated.

2. Send Security Context--Send the business object with the security context credential information, with the connectionId being the same as returned from the stateful operation. This is set in the MO_Instance of this business object:
-->Request processing using Security_Context business object
3. Send Customer Add--Send the business object to add a customer using the same connectionId:
-->Request processing using Customer_Add business object, using the same connectionId provided by the stateful operation request.
4. End Customer Add--End this series of stateful operations using i2SecContext_StatefulRequest by submitting the business object with "Request=End" and the same connectionid.

Specifying business object attribute properties

The i2 connector has various properties that you can set on its business object attributes. This section describes how the connector interprets several of these properties and describes how to set them when modifying a business object.

The following table shows the properties for simple attributes.

Attribute	Description
Name	Unique name of the attribute
Type	All simple attributes should be of type String.
MaxLength	Not used
IsKey	Each business object must have at least one key attribute, which you specify by setting the key property to true for an attribute. The i2 connector does not check for this property.
IsForeignKey	Not used.
Is Required	Set to true if the attribute must have a value in the outgoing XML message.
AppSpecInfo	Not used
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. Rule: You must set and use the default value for the attributes of the polling metaobjects and MO_Instance metaobject. If the default value is set for the instanceId, and no value is set in the incoming business object, the connector takes the default value and tries to connect with this instance.

The following table shows the properties for child object attributes.

Attribute	Description
Name	Name of the child object.
Type	Business object type for the child.
Contained ObjectVersion	For all attributes that represent child business objects, this property specifies the child's business object version number.

Attribute	Description
Relationship	If the child is a container attribute, this is set to Containment.
IsKey	Not used
IsForeignKey	Not used.
Is Required	For relationship details between XML elements and requiredness, see Chapter 3, "XML data handler," in the <i>Data Handler Guide</i> .
AppSpecInfo	For information on this property, see "Identifying business object application-specific information" on page 23,
Cardinality	For relationship details between XML elements and cardinality, see Chapter 3, "XML data handler," in the <i>Data Handler Guide</i> .

Special attribute values

Simple attributes in business objects can have the special value, `CxIgnore`. When it receives a business object from the integration broker, the connector ignores all attributes with a value of `CxIgnore`. It is as if those attributes were invisible to the connector. No XML is generated for them.

Because the i2 connector requires at least one primary key attribute to create a business object, you need to ensure that business objects passed in to the connector should have at least one primary key that is not set to `CxIgnore`.

Additionally, The i2 connector assumes that no attribute of business object *type* has a value of `CxB1ank`. Simple (String) attributes with a value of `CxB1ank` are included in an XML document. Empty double quotation marks ("") in an XML document are used as the PCDATA equivalent of `CxB1ank`.

Identifying business object application-specific information

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

Application-specific information at the business object- level

The following table provides application-specific information at the business object-level for the wrapper business object supported by the i2 connector.

Parameter	Description
Port=	The name of the i2 port type for the operation.
Type=Stateful	A metaobject that signals the beginning or end of a series of stateful requests.

Application-specific information at the attribute level

The following table provides application-specific information at the attribute level for the wrapper business object supported by the i2 connector.

Parameter	Description
Type=	The type represented by the attribute at the wrapper business object attribute level. The type could be input or output representing the input and output for the operation.

Chapter 4. Generating business objects using the i2 ODA

This chapter describes the i2 ODA, an object discovery agent (ODA), which, working with the XML schema ODA, generates business objects for the IBM WebSphere Business Integration Adapter for i2.

This chapter contains the following sections:

- “Overview of the i2 ODA” on page 25
- “Installing the i2 ODA” on page 25
- “Using the i2 ODA in Business Object Designer” on page 27

Overview of the i2 ODA

The i2 Object Discovery Agent (ODA) is a utility to use to obtain the specifications for the i2 business object from the metadata information in i2’s CIS registry. The Business Object Development wizard automates the process. You can view or make modifications to the business object before saving it to the server.

The process for generating an i2 business object has three distinct stages:

1. Identifying the ports, operations, and types for which the i2 ODA generates the schema files; generating the XML schema for the types; and generating the wrapper business object representing the operations with the types as attributes.
2. Processing the i2 generated XML schema files and converting the XML schema to the actual business object for the type.
3. Prior to saving the wrapper business object, saving to the repository the MO_Instance business object and the business objects that were generated for the types using the XML ODA.

For details, see “Steps for using the i2 ODA” on page 27.

Installing the i2 ODA

This section describes how to install and launch the i2 ODA, run multiple instances of the i2 ODA, and work with error and trace message files.

Steps for installing the i2 ODA

Before you begin: This chapter assumes you have already installed the i2 connector, as well as the required software for using the connector (see Chapter 2, “Installing and configuring the connector,” on page 7). Be sure you are using the i2 application version 6.0.1 and the i2 ODA 1.2.x.

To install the i2 ODA, use the Installer for IBM WebSphere Business Integration Adapters. Follow the instructions in the *System Installation Guide for UNIX or for Windows* (for ICS), *Implementing Adapters with WebSphere MQ Integrator Broker*, or *Implementing Adapters with WebSphere Application Server*. When the installation is complete, the following files are installed in the directory on your system where you have installed the product:

- ODA\i2\i2ODA.jar
- ODA\messages\i2ODAAgent.txt

- ODA\i2\start_i2ODA.bat (Windows only)
- ODA/i2/start_i2ODA.sh (UNIX only)

Notes:

1. Except as otherwise noted, this document uses backslashes (\) as the convention for directory paths. For UNIX installations, substitute slashes (/) for backslashes.
2. All product path names are relative to the directory where the product is installed on your system.

Other installation requirements

- i2 provides the MetadataService adapter to obtain the metadata information from the registry. You need to install this adapter on an instance of the i2 application. Be sure to start the adapter prior to using the MetadataService.
- The bindings file for the port MetadataService, for example, TDMMetadata.xml, which contains the port, operation, and type information, needs to be in the i2 configuration directory.

Launching the i2 ODA

Before you begin: Ensure that the i2 ODA and XML schema ODA are installed on your system.

To launch the i2 ODA, run the appropriate file:

- **UNIX:** start_i2ODA.sh
- **Windows:** start_i2ODA.bat

You have to add the proper path to the start files for CIS-SDK and j2ee.jar. You need to point to the exact location of connector jar as found in the i2 lib directory.

Example: The following path information needs to be added to the start_i2.bat file:

```
set I2_CIS_HOME_DIR=C:\i2\CIS\6.0\cis-sdk
set J2EE_PATH=C:\J2EE_JAR
set CONNECTORJAR=
```

Note: These are just examples. You should change the path information depending on your local installation.

You configure and run the i2 ODA using Business Object Designer. Business Object Designer locates each ODA by the name specified in the *AGENTNAME* variable of each script or batch file. The default ODA name for this connector is *i2ODA*.

Working with error and trace message files

Error and trace message files (the default is *i20DAAgent.txt*) are located in \ODA\messages\, which is under the product directory. These files use the following naming convention:

```
AgentNameAgent.txt
```

Example: If the *AGENTNAME* variable specifies *i2ODA1*, the tool assumes that the name of the associated message file is *i20DA1Agent.txt*

You can have a message file for each ODA instance or have differently named ODAs use the same message file. The name of the message file is specified in Business Object Designer as part of the ODA configuration.

Note: Failing to correctly specify the message file's name when you configure the ODA causes it to run without messages. For more information on specifying the message file name, see "Configure agent properties" on page 28.

During the configuration process, you specify:

- The name of the file into which the i2 ODA writes error and trace information
- The level of tracing, which ranges from 0 to 5

The following table describes the tracing levels.

Trace Level	Description
0	<ul style="list-style-type: none">• Logs errors and fatal errors from the i2 ODA application• Logs warnings that require a system administrator's attention
1	Traces all entering and exiting messages for method
2	Traces the ODA's properties and their values
3	Traces the names of all business objects
4	Traces business object properties and the values received
5	<ul style="list-style-type: none">• Indicates the ODA initialization values for all of its properties• Traces the business object definition dump

For information on where to configure these values, see "Configure agent properties" on page 28.

Using the i2 ODA in Business Object Designer

This section describes how to use the i2 ODA in Business Object Designer to generate business objects. For information on launching Business Object Designer, see the *IBM WebSphere Business Integration Adapters Business Object Development Guide*.

After you launch an ODA, you must launch Business Object Designer to configure and run it. Business Object Designer provides a wizard that guides you through six steps to generate a business object definition using an ODA. The six steps are as follows:

1. Select the agent.
2. Configure the agent properties.
3. Expand the nodes and select the port types, operations, and input/output types.
4. Confirm the selection, generate the wrapper business objects, and save.
5. Complete the business object and generate the business objects for the types.
6. Save the business object files.

Details for each step follow.

Steps for using the i2 ODA

Before you begin: You need to start the i2 Business Object Designer wizard.

1. Open Business Object Designer.
2. From the File menu, select New Using ODA....

Result: Business Object Designer displays the first window in the wizard, named Select Agent.

Perform the following steps:

Select the agent

To select the ODA:

1. Click Find Agents to display all registered or currently running ODAs in the Located agents field.
2. Select the desired ODA from the displayed list.

Note: The ODA for i2 has a default name of *i2ODA*. The agent name depends on the value of the *i2* variable in the *start_i2ODA.bat* or *start_i2ODA.sh* file.

Recommendation: When you run multiple instances of the ODA utility, you should change the default name by creating a separate batch file for each instance or by specifying a unique name in the *AGENTNAME* variable of each batch file.

When multiple instances of the ODA are running on different machines, they are visible in the Business Object Designer by their *i2* ODA values. If two ODAs have the same *i2* value, then either of the ODAs can be used, with possibly undesirable results. You can assign unique names to such ODAs by prefixing the *i2* name with the host machine name, or by using an ORB finder (for example, *osfind*) to locate existing CORBA object names on your network.

Result: Business Object Designer displays your selected agent in the **Agent's name** field.

Configure agent properties

The first time Business Object Designer communicates with the *i2* ODA, it prompts you to enter a set of initialization properties. You can save these properties in a named profile so that you do not need to re-enter them each time you use the *i2* ODA. For information on specifying an ODA profile, see the *Business Object Development Guide*.

In addition to configuring these one-time properties, you need to configure properties to connect to the CIS agent and to define the tree nodes.

The following table describes the properties to configure.

Row number	Property name	Property type	Description
1	DefaultBOPrefix	String	Text that is prepended to the name of the business object to make it unique.
2	SchemaFileLocation	String	Example: <i>i2_BO</i> Path where the generated.xsd files are stored. This is mandatory; you must specify the path to store the schema files.
3	MessageFile	String	Path to the error and message file. If the file is not specified, the error messages from the ODA are not displayed. The <i>i2</i> ODA displays the file name according to the naming convention. Example: If the agent is named <i>i2ODA</i> , the value of the message file property displays as <i>i2ODAAgent.txt</i> .

Row number	Property name	Property type	Description
4	CISAgentHostName	String	Host name of the CIS agent, it is specified if the CIS agent is running on a remote machine.
5	MetadataService	String	Name of the i2 CIS port where the MetadataService is located. The default is MetadataService.
6	PortTypesOperation	String	Name of the operation to get the port types through metadata introspection. The default is getPortTypes.
7	SchemasOperation	String	Name of the operation to get the schemas for the selected operations. The default is getSchemasForOperation.

You can also configure several optional parameters for the agent when it starts up.

- **TraceFileName**--File into which the i2 ODA writes trace information. The command line option for this parameter is `-t`. The i2ODA names the file according to the naming convention.

Example: If the agent is named `i20DA`, it generates a trace file named `i20DAtrace.txt`.

- **TraceLevel**--Level of tracing enabled for the i2 ODA. For more information, see "Working with error and trace message files" on page 26.

Expand nodes and select port types, operations, and input/output types/formats

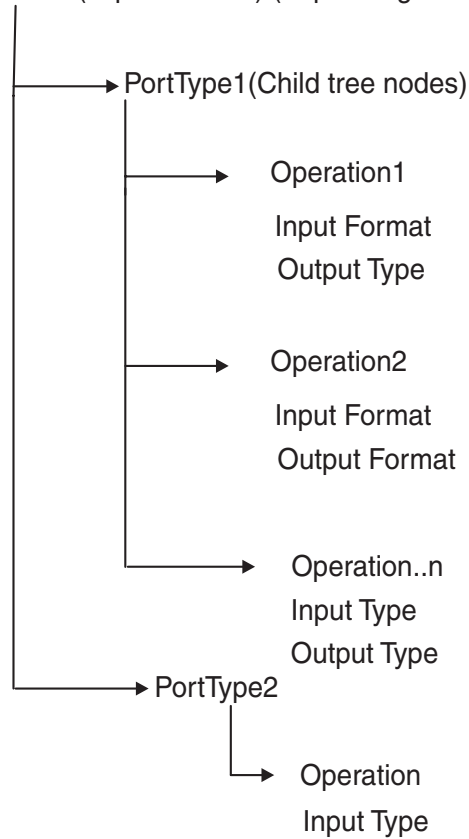
The i2 ODA uses the properties you configured in the preceding step to connect to the specified i2 application. Business Object Designer displays the metadata information about the registered ports, operations for each port, and the input and output types for each operation as a tree structure. You can expand the tree nodes for the ports and operations by right-clicking on a node. The *type* node is the leaf node of the tree, so it is not expandable.

A distinction is made between "Types" and "Formats" when dealing with the input and output values for an operation. When displaying the input and output business objects for an operation, if it is a "Format" that will be displayed instead of a "Type," it will display as "Output Format" instead of "Output Type".

Note: In this document, when we say "Type" we mean "Type" or "Format".

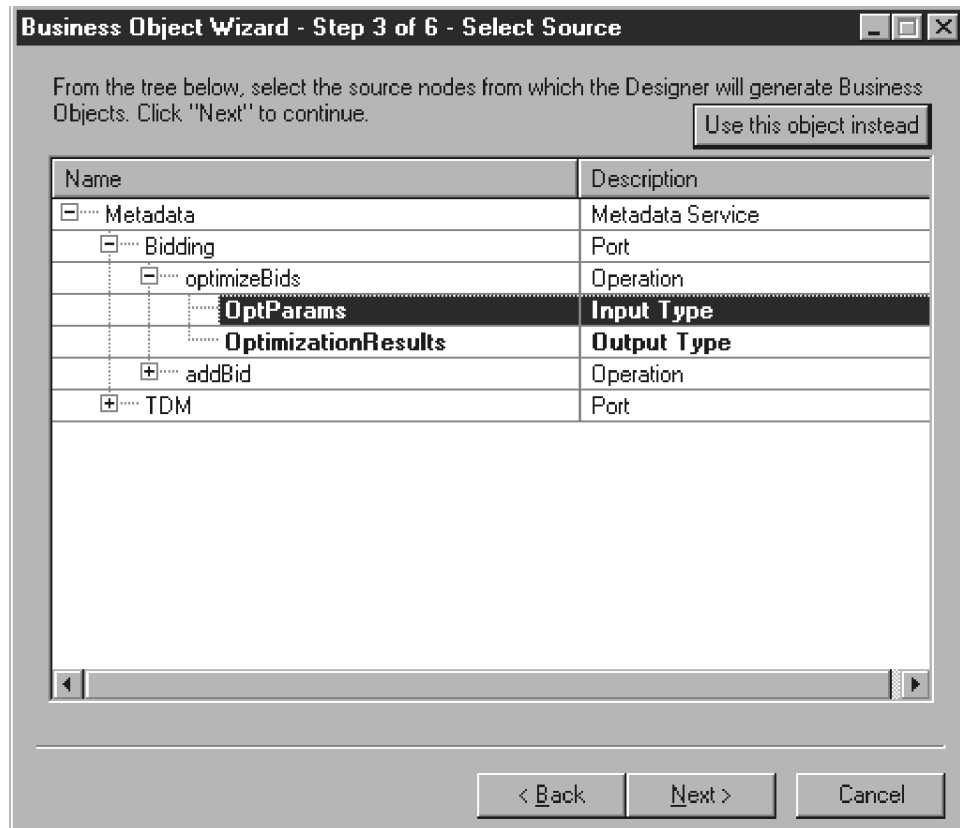
The following diagram shows a tree view with expanded nodes.

Metadata (Top tree node) (Expanding Metadata lists all the port types)



- Select the port, operation, and type for generating the business object. Each operation has an input and output type on a port, and each of these types needs

to have a corresponding business object.



Result: The i2 ODA will create the schema file for the chosen type. The schema files form the input for the XML schema ODA to generate the corresponding business objects. The XML schema file name follows the naming convention, input/output_operation_type.

Example: For the persistOrder operation, both the input type and output type is Order, but the formats for the two are different. The i2 ODA validates these and generates different schema files for the input and output types:

- i2BO_in_persistorder_Order.xsd (input)
- i2BO_out_persistorder_Order.xsd (output)

The schema files are stored in the directory specified by the SchemaFileLocation property for the ODA. The i2 ODA checks the schema location for the existence of the schema prior to creating the new one. If there already exists an xsd for the type, the i2 ODA does not duplicate the schema.

Confirm selection, generate wrapper business object, and save

1. Confirm the operations you have selected for the i2 ODA to generate wrapper business objects.

Result: The i2 ODA creates a wrapper business object to represent the operation with the chosen port type. Each operation has one wrapper business object. The input and output types of the operation form the attributes of this wrapper business object, with the operation as the verb. The port type becomes the application-specific information for this business object.

The name of the wrapper business object is DefaultBOPrefix_operation. The wrapper business object contains:

- Port information
- MO_instance containing:
 - InstanceId
 - ConnectionId
 - Username
 - Password
- Dummy key (as the business object creation will fail without a key)
- Two single cardinality attributes representing the input and output types for the operation. The attributes are named as BOPrefix_in_operation_type and BOPrefix_out_operation_type.

Example: The wrapper BO for the operation persistOrder is i2BO_persistOrder.

i2bo_persistorder Port=TDM
MO_Instance InstanceId=prapulla2k ConnectorId=i2Connector Username=i2User Password=i2Password
DummyKey
i2bo_in_persistorder_order Type=Input
i2bo_out_persistorder_order Type=Output

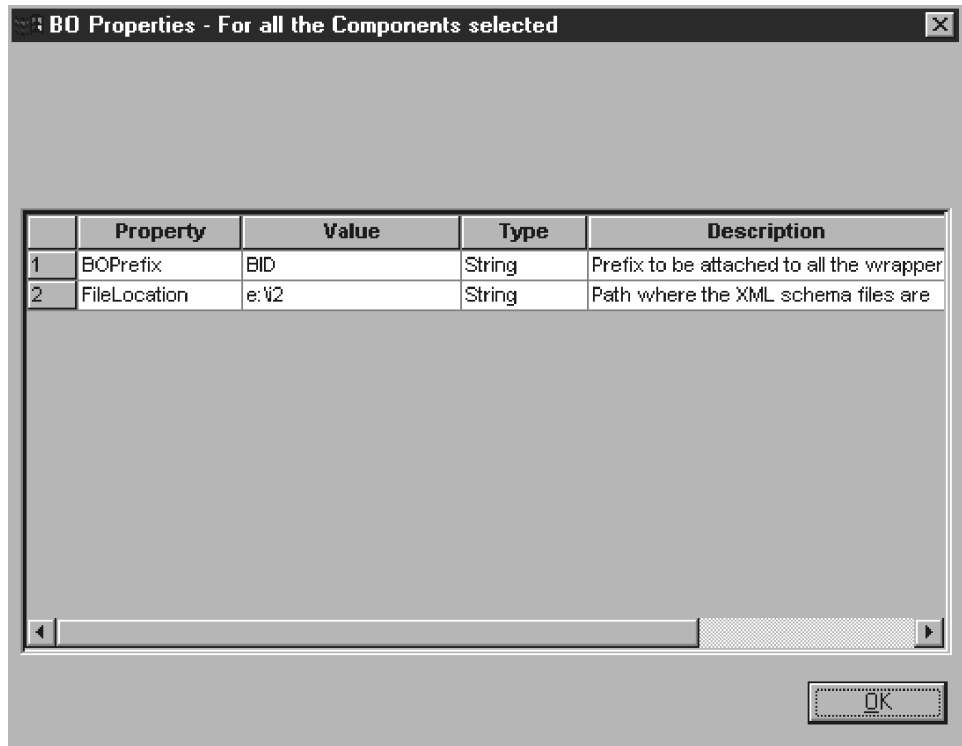
2. Save the wrapper business object to a file. Saving it to the InterChange Server will fail because the corresponding dependent attribute business objects have not yet been created by the XML ODA schema.

Guideline: When you save the business objects into a file in this step, the current Business Object Designer wizard looks at the machine where the agent is running. Another pop-up window prompts you for a location to save the generated wrapper business objects.

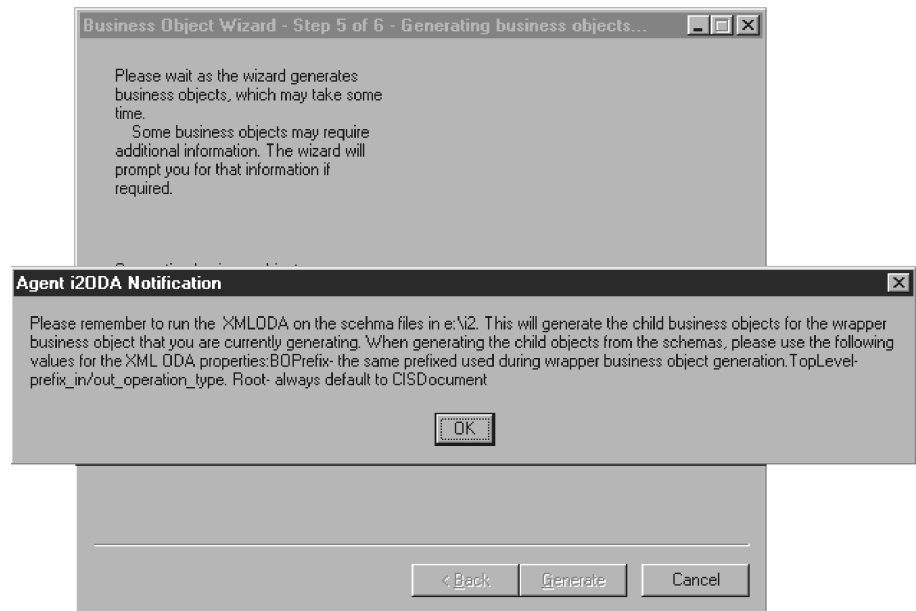
Complete the business object and generate the business objects for the types

1. Check the property values that Business Object Designer displays in the BO Properties window. If you are satisfied with the value you previously entered in the Configure Agent window (for example, for DefaultBOPrefix), you do not

need to change the value here.



2. Be sure the XML schema ODA agent is running.
 - a. The XML schema ODA reads the schema files previously saved to the SchemaFileLocation and generates the business objects for the input and output types.



- b. Save these business objects to the same directory as the wrapper business objects.

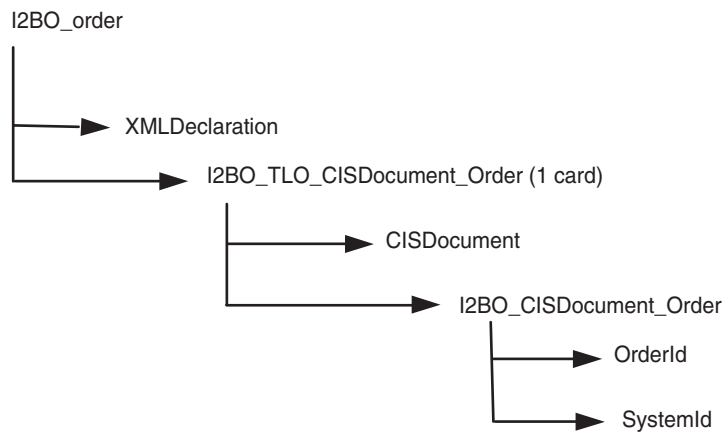
Guideline: Remember to run the XML ODA on the schema files in the directory you specified when running the ODA. This will generate the child

business objects for the wrapper business object that you generated. When generating child objects from the schema, use the following values for the XML ODA properties:

- BOPrefix--This should be the same prefix used during the wrapper business object generation.
- TopLevel--This should be prefix_in/out_operation_type.
- Root--This should always default to CISDocument.

BOPrefix for the XML schema ODA should be the same as the BOPrefix for the i2 ODA.

The following diagram shows the business object that the XML schema ODA generates for i2_order.xsd. The XML data handler uses the combination of the element next to CISDocument and BOPrefix to get the business object name.



Note: The inclusion of an `<xsd:any minOccurs="0" maxOccurs="unbounded" namespace="##any" processContents="skip"/>` tag, will no longer cause the business object generation to fail. For more information see the *Data Handler Guide*.

Save the business object files

Now that all the required business objects are generated, you need to save them to the InterChange Server for use by the collaborations. Use the Business Object Designer utility to copy the business objects to the server. You can concatenate the files together into a single file and copy them to the server.

Guideline: Be sure to run the XML schema ODA prior to saving the wrapper business objects to the server.

Example:

Property	Value
FileName	D:\i2\odaschema\i2persist_in_persistOrder_Order_Order.xsd
Root	CISDocument
TopLevel	in_persistOrder_Order
BOSelection	false
BOPrefix	i2persist

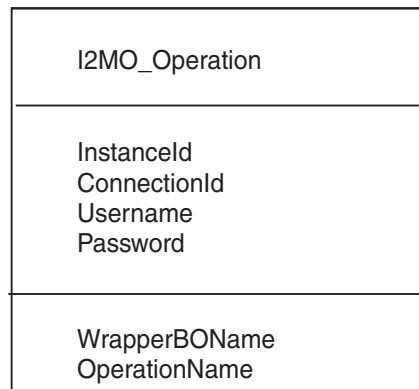
Property	Value
Doctypeor SchemaLocation	true
TraceFileName	XMLODAtrace.txt
TraceLevel	5
MessageFile	XMLODAAgent.txt

At this stage, you can decide whether to run different operations on different instances. You can clone the MO_Instance and set a default value for the instanceId on these. You will need to replace the default MO_Instance with the newly created ones in the wrapper business objects for the operations.

Create the metaobject for polling

Once the business objects are created, you need to create the metaobjects for polling using System Manager. These objects shall have the i2MO prefix followed by the operation. The attributes need to have a default value. This information is used during polling to register the specific operation and check the output from the i2 applications for the registered operation.

The following diagram shows the structure of the i2 metaobject for i2 MO_Operation. The diagram shows the attributes of the metaobject--the instanceId, connectionId, username, and password--and the operation name and business object wrapper name.



For more information on configuring metaobjects, see Chapter 3, "Understanding business objects for the connector," on page 15.

Chapter 5. Troubleshooting and error handling

This chapter describes how the i2 connector for i2. handles errors. The connector generates logging and tracing messages. The chapter contains the following sections:

- “Logging error messages” on page 37
- “Tracing messages” on page 40
- “Running the adapter against a CIS on a different subnet” on page 41
- “Tips for troubleshooting” on page 42

Logging error messages

The connector logs an error message whenever it encounters an abnormal condition during processing, regardless of the trace level. When such an error occurs, the connector also prints a textual representation of the failed business object as it was received. It writes the text to the i2 Adapter log file, whose file name corresponds to the connector property LogFileName.

The message contains a detailed description of the condition and the outcome and may also include extra information that may aid in debugging, such as business object dumps or stack traces (for exceptions).

Error types: Error messages are of two types:

- Errors--conditions that the connector can recover from, usually by abandoning the current processing.
- Fatal errors--unrecoverable error conditions. See the following sections on polling-related and request processing errors.

Structure of error messages

All the error messages that the connector generates are stored in a message file named `i2Adapter.txt`. Each error message typically has a message ID, the error message, and an explanation section for a detailed description and tips to rectify the problem.

Message ID

Message

[EXPL]

The message IDs for the i2 connector range from 90000 to 92000, with 90000 to 91000 set aside for polling-related error messages, and 91000 to 92000 set aside for request processing error messages.

Example: The following exemplifies the message structure, where `nnnnn` represents the message ID.

```
nnnnn
Not able to get a connection for this instance {1}.
[EXPL]
Please ensure that the instance specified is up.
{1}--Parameter to the error message, in this case the instance id.
```

Polling-related error messages

The following table describes polling-related error messages. These are logged in the i2 Adapter log file.

Notes:

1. In some cases, the connector logs a fatal error (log message type of XRD_FATAL) so that e-mail notification can be triggered. For logging this error with the integration broker, you need to set the connector property LogAtInterchangeEnd to true.
2. E-mail notification will be sent only if the e-mail connector is configured.

Error description	Error type	Handling by i2 connector
Connection lost while polling	Fatal error	The connector detects the connection error at the time of a poll call. It logs a fatal error (log message type of XRD_FATAL) and dumps the XML message. Fatal error can trigger e-mail notification. For logging this error with the integration broker, you need to set the connector property LogAtInterchange to true and continue with polling on other operations that might be running on instances that are active.
Not able to register an operation with the CIS Agent	Fatal error	The connector logs a fatal error (log message type of XRD_FATAL) and dumps the XML message. Fatal error can trigger e-mail notification. For logging this error with the integration broker, you need to set the connector property LogAtInterchange to true. With subsequent poll calls, the i2 connector tries to re-register the operations that could not be registered with the previous poll call. In case registering all the operations fail, we assume that the CIS agent is down; the connector returns APPRESPONSETIMEOUT, which shuts down the i2 connector.
No metaobjects configured for polling	Error	The i2 connector always returns a FAIL after logging that the metaobjects are not configured for polling.
Default value for the metadata object attributes not set	Error	For details on the default value attribute property, see Chapter 3, "Understanding business objects for the connector," on page 15. The error "required information for polling was not found for the specified metaobject" is logged and the processing continues for other messages.

Error description	Error type	Handling by i2 connector
Not able to fetch the message from CIS Agent	Error/SUCCESS	In case the poll call to the i2 application returns a null, or there are no events for the operation, the connector continues to poll for other operations. In case an exception is caught from any CIS Client API, an error message containing the word ERROR_PROCESSING_EVENT is logged. The connector continues to poll for other operations.
Fail to convert XML message to IBM business object	Error	The error that the XML message has a syntax error is logged for the message, The XML message gets dumped to the log file, and the processing continues for other messages.
Any error when posting event to the broker	Error/Fatal	In case of a return code of CONNECTOR_NOT_ACTIVE, the connector logs a fatal error and returns APPRESPONSETIMEOUT which shuts down the connector. The error is logged with the status of ERROR_POSTING_EVENT. In case of a return code of NO_SUBSCRIPTION_FOUND, the error is logged with the status of UNSUBSCRIBED and polling continues. For a return code of FAIL, the error is logged with the status of ERROR_POSTING_EVENT for the business object, and the polling continues for the other messages.
Event not subscribed to	Error	The error is logged with the status of UNSUBSCRIBED for the business object, the XML message is dumped to the log file, and the polling continues for the other messages.

Service call request processing error messages

The following table describes service call request processing error messages. These are logged in the i2 Adapter log file.

Error description	Error type	Handling by i2 connector
Not able to get a connection for the specified port type and instance.	Fatal error	The connector detects the connection error at the time of a service request processing call. It logs a fatal error with the status of FAIL in the exception and a detailed exception message stating the cause of the exception set on the exception object.

Error description	Error type	Handling by i2 connector
No instanceId found in the metadata object within the incoming business object	Error	The connector checks for the default value setting for the instanceId attribute of the metaobject within the wrapper business object. If set, the connector tries to connect to this instance and execute the operation. If there is no default value, it logs the error message with the status of FAIL and a detailed exception message stating the cause of the exception set on the exception object.
Not able to convert the incoming child business object attributes to XML.	Error	The i2 connector logs the message to the adapter log and sets the status on the exception to FAIL.
Failure executing the operation on the i2 side.	Error	The i2 connector logs the message from the Resource Exception to the adapter log and sets the status on the exception to FAIL.
Not able to convert the returned CIS Record if the operation was SUCCESSFUL to XML.	Error	The i2 connector logs the message to the adapter log and sets the status on the exception to FAIL.
Not able to convert the XML message to the business object.	Error	The i2 connector logs the message to the adapter log and sets the status on the exception to FAIL. It also dumps the XML message to the log file along with the error message.
Execute method returns null output	Error/Success	In case the operation does not have an output type, the execute method execution is considered a SUCCESS. If an output is expected, the execute method issues an exception which is caught by the connector.

Tracing messages

Tracing is an optional debugging feature you can turn on to closely follow a connector's behavior. Tracing messages are configurable and can be changed dynamically. You set various levels depending on the desired detail. Trace messages, by default, are written to STDOUT (screen). You can also configure tracing to write to a file.

Recommendation: Tracing should be turned off on a production system or set to the lowest possible level to improve performance and decrease file size.

The following table describes the types of tracing messages that the i2 connector outputs at each trace level. All the trace messages appear in the file specified by the connector property TraceFileName. These messages are in addition to any tracing messages output by the IBM WebSphere Business Integration Adapter architecture.

Tracing Level	Tracing Messages
Level 0	Message that identifies the connector version. No other tracing is done at this level. This message is always displayed. Example: '2002/07/10 15:01:46.812: This is version 1.0 of the i2 Adapter'.
Level 1	Messages delivered each time the pollForEvents method is executed.
Level 2	<ul style="list-style-type: none"> • Messages logged each time a business object is posted to the integration broker from gotAppEvent. • Messages that indicate each time a business object request is received.
Level 3	Not applicable
Level 4	<ul style="list-style-type: none"> • Application-specific information messages, for example, messages showing the values returned by the functions that parse the business object's application-specific information fields. • Messages that identify when the connector enters or exits a function, which helps trace the process flow of the connector.
Level 5	<ul style="list-style-type: none"> • Messages that indicate connector initialization, for example, messages showing the value of each configuration property retrieved from the integration broker. • Messages that comprise a business object dump. At this trace level, the connector outputs a textual representation of the business object it is processing before it begins processing the object (showing the object the connector receives from the collaboration), and after it is done processing the object (showing the object the connector returns to the collaboration).

Running the adapter against a CIS on a different subnet

When the CIS agent and CIS adapters are running on a different subnet from the Adapter for i2, you need to tell the CIS agent and CIS adapter the fully qualified name or IP address that the RMI Server Hostname is using.

You will need to set the JVM argument when starting the CIS agent and any installed adapter.

Perform the following steps to set the instance ID and RMI Server Hostname.

Note: Henceforth, Hostname is the machine name or the IP address of the machine that the CIS agent and CIS adapters are running. You can use the IP address if it will not change.

1. If you have installed the adapter, uninstall it by running `uninstallAdapter.py`.
2. Change the Metadata file `Bindings.xml` for the CIS adapter by inserting `<bindings instanceID=Hostname>` as an attribute to the `Bindings` element.
3. Reinstall the adapter with this JVM argument:

```
installAdapter.py -j-Djava.rmi.server.hostname=<hostname> -m
<metadatabinding file>
```
4. Start the CIS agent giving the same JVM argument:

```
-j-Djava.rmi.server.hostname=<hostname>
```
5. Restart the CIS agent stating the RMI Server Hostname as a JVM argument property.

6. Start the newly configured adapter stating the RMI Server Hostname as a JVM argument property.
7. To enable the CIS agent to send event notifications to the machine that the Adapter for i2 is running, you need to tell it the fully qualified hostname of the machine where the Adapter for i2 is running. To do this, add the line:
`MACHINE_NAME=mymachine.company.com`

to the file `[i2_CIS_install]/{cis-sdk}/properties/cisclient.properties` where, `mymachine.company.com` is the fully qualified name of the machine where the Adapter for i2 is running. This tells the CIS Client (Adapter for i2) to use the machine name specified in the `cisclient.properties` file as its machine name when running remotely.

8. In some cases, you will need to state the machine name of the CIS agent when the machine name of the CIS agent cannot be resolved from a remote machine running the Adapter for i2. To do this:
 - a. Set the IP address of the CIS agent machine as the value in the `<MACHINE_NAME>` element in the `settings.xml` file for your CIS agent.
 - b. Run `configure.py` to make the changes.
 - c. Restart the agent and all adapters.

For more information on running the CIS on different subnets, see your CIS guide.

Tips for troubleshooting

Use the following tips for troubleshooting problems:

- If the CIS agent is running remotely, try to ping the remote machine and also ping this machine from the remote machine.
- Check the CIS agent service.
- Check that the adapters are running.
- Be sure the business object structure is consistent with the operation.
- See if any default value is set for the `instanceId` like in the metaobjects.
- If you want to run the connector in request process mode only, be sure you start the connector with the `-fno` option set.

Appendix A. Standard configuration properties for connectors

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

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

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

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

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

New and deleted properties

These standard properties have been added in this release.

New properties

- XMLNamespaceFormat

Deleted properties

- RestartCount

Configuring standard connector properties

Adapter connectors have two types of configuration properties:

- Standard configuration properties
- Connector-specific configuration properties

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

Using Connector Configurator

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

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

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 2 on page 45 below.

Summary of standard properties

Table 2 on page 45 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 2 on page 45, 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 2. 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 2. 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 2. 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 2. 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>
WsifSynchronousRequest Timeout	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 appendix on Connector Configurator.

ConcurrentEventTriggeredFlows

Applicable only if RepositoryDirectory is <REMOTE>.

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

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

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

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

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

ContainerManagedEvents

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

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, see the appendix on Connector Configurator.

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

<http://www.ibm.com/software/websphere/wbiadapters/infocenter>, or
<http://www.ibm.com/websphere/integration/wicsserver/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 50.

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.

WsifSynchronousRequest Timeout

WAS integration broker only.

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

The default value is 0.

XMLNameSpaceFormat

WebSphere message brokers and WAS integration broker only.

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

The default value is short.

Appendix B. Connector Configurator

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

You use Connector Configurator to:

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

Note:

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

The topics covered in this appendix are:

- “Overview of Connector Configurator” on page 61
- “Starting Connector Configurator” on page 62
- “Creating a connector-specific property template” on page 63
- “Creating a new configuration file” on page 65
- “Setting the configuration file properties” on page 68
- “Using Connector Configurator in a globalized environment” on page 74

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 62).

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 63 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 67.)

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 63.
- 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 70..
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 44.

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>
```

Appendix C. 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