IBM WebSphere Business Integration Adapters

Adapter for i2 User Guide

Adapter Version 14.x

IBM WebSphere Business Integration Adapters

Adapter for i2 User Guide

Adapter Version 14.x

Note!
Before using this information and the product it supports, read the information in Appendix E, "Notices," on page 97.

13September2005

This edition of this document applies to IBM WebSphere Business Integration Adapter for i2 (5724-G91), version 1.4.x.

To send us your comments about IBM WebSphere Business Integration documentation, e-mail doc-comments@us.ibm.com. We look forward to hearing from you.

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

Contents

About this document

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.

What this document includes

This document describes installation, connector property configuration, business object development, and troubleshooting for the IBM WebSphere Business Integration Adapter for i2.

What this document does not include

This document does not describe deployment metrics and capacity planning issues, such as server load balancing, number of adapter processing threads, maximum and minimum throughputs, and tolerance thresholds.

Such issues are unique to each customer deployment and must be measured within or close to the exact environment where the adapter is to be deployed. You should contact your IBM services representative to discuss the configuration of your deployment site, and for details on planning and evaluating these kinds of metrics, given your specific configuration.

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:

ProductDir	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.
italic, *`italic`*	Indicates a new term the first time that it appears, a variable name, or a cross-reference.
blue outline	A blue outline, which is visible only when you view the manual online, indicates a cross-reference hyperlink. Click inside the outline to jump to the object of the reference.
`{ }`	In a syntax line, curly braces surround a set of options from which you must choose one and only one.
`|`	In a syntax line, 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, `option[,...]` 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 `<server_name><`*`connector_name`*`>tmp.log`.
`/, \`	In this document, backslashes (\) are used as the convention for directory paths. For UNIX installations, substitute slashes (/) for backslashes. All product path names are relative to the directory where the connector for i2 is installed on your system.
`%text%` and *`$text`*	Text within percent (%) signs indicates the value of the Windows `text` system variable or user variable. The equivalent notation in a UNIX environment is *`$text`*, indicating the value of the *`text`* UNIX environment variable.

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 1.4.x

- The adapter for i2 now supports i2 Infrastructure Services (CIS) version 6.1.
- Username and password set in the adapter configuration properties are used to authenticate users when JAAS authentication is enabled for i2 CIS Adapters.
- The adapter for i2 now supports durable subscriptions for event notification. The adapter can be configured for durable subscription. The UnregisterOperations and Unregister connector configuration properties and the polling meta-object property define whether or not the adapter should unregister its durable subscription.
- The adapter for i2 can now redeliver messages that fail to reach the integration broker. This is configurable at the polling meta-object level with the new property AutoAcknowledge.
- The adapter for i2 will now clean-up closed connections, removing them from the persistent store.

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 6.1 SDK CIS-enabled i2 application. Many i2 modules versions 5.2 and above, support version 6.1 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

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

CIS includes three primary components:

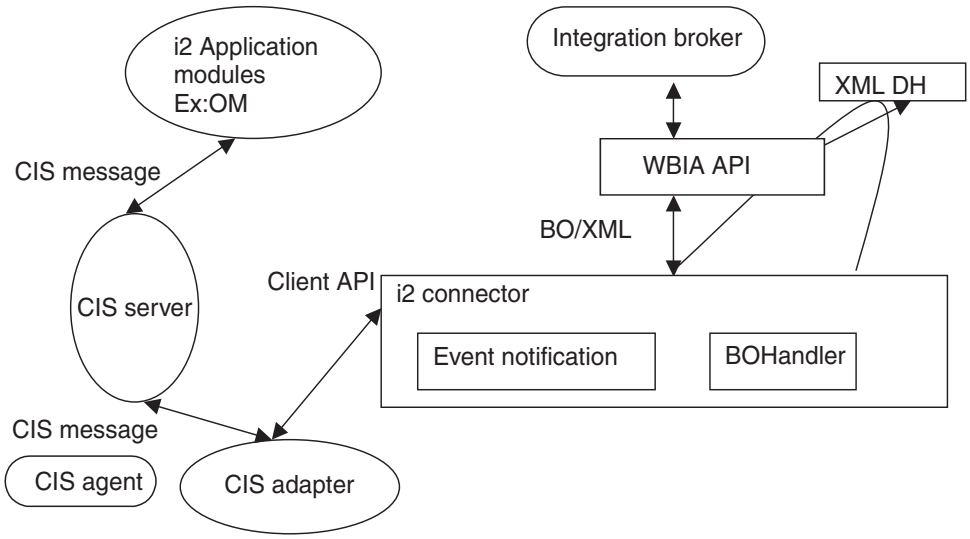
- CIS Front Bus--used by applications to specify an XML metadata format interface that details the available functions and their expected input and output data. CIS scripts generate the required representations of input and output data in an XML schema and Java Beans. Product teams implement these functions 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 19.

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 19.

Common Event Infrastructure

This adapter is compatible with IBM's Common Event Infrastructure, a standard for event management that permits interoperability with other IBM WebSphere event-producing applications. If Common Event Infrastructure support is enabled, events produced by the adapter can be received (or used) by another Common Event Infrastructure-compatible application.

For more information refer to the Common Event Infrastructure appendix in this guide.

Application Response Measurement

This adapter is compatible with the Application Response Measurement application programming interface (API), an API that allows applications to be managed for availability, service level agreements, and capacity planning. An ARM-instrumented application can participate in IBM Tivoli Monitoring for Transaction Performance, allowing collection and review of data concerning transaction metrics.

For more information refer to the Application Response Measurement appendix in this guide.

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.

Error messages, if any, are logged to the i2 adapter log file. For more information, see Chapter 5, "Troubleshooting and error handling," on page 41.

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 19 and Chapter 4, "Generating business objects using the i2 ODA," on page 29.

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, connectionId, and AutoAcknowledge attributes 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 (if it did not unregister). This can be referred to as durable subscription. Durable subscription is maintaind if the adapter is not configured to unregister for operations at the connector level or at the polling meta-object level.

   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 9.
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 19.
6. The connector sends the wrapper business object to the integration broker for further processing.

   i2 CIS SDK version 6.1 contains a feature called Interaction Acknowlegement that enables the redelivery of messages that were not confirmed to be delivered to the integration broker. It does this by allowing a client to state whether or not messages are auto-acknowleged and retaining messages on the i2 Message Bus until they have been successfully delivered. This feature can be enabled by setting the AutoAcknowlege property for polling meta-objects to `false`.

   If the delivery of a message fails to be delivered to the broker and the Auto Acknowlege field is set to `false`, the following happens (depending on the status returned by the `gotApplEvents` method:

   - Fail - The adapter will leave the message on the bus, to be redelivered at a later time.
   - No Subscription Found - The adapter will leave the message on the bus, to be redelivered at a later time.
   - Connector Not Active - The adapter will leave the message on the bus, to be redelivered at a later time.

Note: If AutoAcknowledge is set to `false`, the connectionId attribute is mandatory and must not be re-used by any other polling meta-object. The default behavior is AutoAcknowlege `true`.

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 19, and Chapter 4, "Generating business objects using the i2 ODA," on page 29.

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 23. For information on configuring the InteractionResolutionAttempts and ExecutionTimeout properties, see Chapter 2, "Installing and configuring the connector," on page 9.

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 for i2 environment

Broker compatibility

This adapter runs with the WebSphere Business Integration Adapter FrameworkV2.6 and requires one of the following:

- WebSphere InterChange ServerV4.2.2,V4.3
- WebSphere MQ IntegratorV2.1
- WebSphere MQ Integrator BrokerV2.1
- WebSphere Business Integration Message BrokerV5.0.1
- WebSphere Application Server EnterpriseV5.0.2, in conjunction with WebSphere Studio Application Developer Integration EditionV5.0.1
- WebSphere Business Integration Server FoundationV5.1.1

Adapter platforms

In addition to a broker, this adapter requires one of the following operating systems:

- All operating system environments require the Java compiler (IBM JDK 1.4.2 for Windows 2000) for compiling custom adapters
- **AIX:**
  AIX 5.1 with Maintenance Level 4
- **Solaris:**
  Solaris 9 (2.9) with Solaris Patch Cluster dated February 11, 2004 or later. This adapter supports 32-bit JVM on a 64-bit platform.
- **HP-UX:**
  HP-UX 11.i (11.11) with June 2003 GOLDBASE11i and June 2003 GOLDAPPS11i bundles
- **Windows:**
  Windows 2000 (Professional, Server, or Advanced Server) with Service Pack 4
  Windows XP with Service Pack 1A, for WebSphere Business Integration Adapter Framework (administrative tools only)

Additional requirements

The WebSphere Business Integration Adapter for i2 also requires the following hardware and software:

- CIS SDK 6.1
- i2 application 5.2 or above
- JDK 1.4.2 compliant connector.jar
- Appropriate CIS adapter (MetadataService adapter is required fo the i2 ODA)
- CIS libraries (cis.jar, cis-sso.jar, cis-sso-spi.jar, cis-util.jar, jdom.jar, jmxri.jar, log4j.jar, jaas.jar, connector.jar)
- IBM WebSphere Business Integration XML datahandler
- IBM WebSphere Business Integration API (WBIA.jar)
- i2 CIS 6.1 SDK properties directory
- If i2 CIS is running with WebSphere MQ bindings, the .bindings file

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 set 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 set. Unicode contains encodings for characters in most known character code sets (both single-byte and multi-byte). 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 *Installing WebSphere Business Integration Adapters* guide 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.
3. 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 *ProductDir*	**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.

Subdirectory of *ProductDir*	Description
connectors\i2\Dependencies	Contains sample files for creating business objects, polling meta-objects, and stateful operations meta-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 71.

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 47.

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	*Unique name specified for each connector*	i2 adapter
ApplicationUserName	*User name for the i2 connection used for authentication*	i2User
ApplicationPassword	*Password for the i2 connection used for authentication*	i2Password
CISAgentHostName	*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.*	String host name **Example:** any machine name like California
ExecutionTimeout	Time in milliseconds before the call to i2 application terminates.	Default is 30000
EventSubscriptionConnectionId	*Default connectionId to be used for request processing*	EventConn1
InteractionResolutionAttempts	*Number of times that an interaction should be attempted to be resolved*	Default is 2
PollQuantity	*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.*	Default is 1
UnregisterOperations	*The global value used to determine whether the adapter should unregister its durable subscriptions at shutdown. This can be overriden at the polling meta-object level.*	Default is true
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.

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 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 13.
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**. On Windows systems the startup script should reside in the connector's runtime directory:

ProductDir`\connectors\`*connName*

where *connName* identifies the connector.

On UNIX systems the startup script should reside in the *UNIX ProductDir*/bin directory.

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`
Windows	`start_connName.bat`

When the startup script runs, it expects by default to find the configuration file in the *Productdir* (see the commands below). This is where you place your configuration file.

Note: You need a local configuration file if the adapter is using JMS transport.

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 -start connName brokerName [-cconfigFile ]`

  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, which is launched when you start System Manager running with the WebSphere Application Server or InterChange Server broker:

  You can load, activate, deactivate, pause, shutdown or delete a connector using this tool.
- From System Manager (available for all brokers):

  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*.

Start scripts

To start the adapter for i2, you may use the following start scripts:

- @echo off
- set local
- REM Set Adapter Specific Variables
  REM set the i2 CIS installation directory here
  REM
  set I2_CIS_HOME_DIR=
- REM set the name to be the appliation connector that is starting
  set CONNAME=%1
  set CONNPACKAGENAME=com.crossworlds.connectors.i2.i2AdapterAgent
- REM set the server name to be the interchange that is being targeted
  setSERVER=%2
- set CW_I2JAR=CWi2.jar
  set I2JAR="%I2_CIS_HOME_DIR%"\lib\cis.jar;"%I2_CIS_HOME_DIR%" \lib\jaas.jar;
  set I2JAR=%I2JAR%;"%I2_CIS_HOME_DIR%"\lib\jmxri.jar; "%I2_CIS_HOME_DIR%"\lib\jaas.jar;
  set I2JAR=%I2JAR%;"%I2_CIS_HOME_DIR%"\lib\cis-sso.jar;"%I2_CIS_HOME_DIR%"\lib\cis-sso-spi.jar;
  set I2JAR=%I2JAR%;"%I2_CIS_HOME_DIR%"\lib\jdom.jar; "%I@_CIS_HOME_DIR%"\lib\cis-util.jar;
  set I2JAR=%I2JAR%;"%I2_CIS_HOME_DIR%"\lib\connector.jar
- set I2PROPERTIES="%I2_CIS_HOME_DIR%"\properties;
- REM End Adapter Specific Tasks
- REM IF WBIA_RUNTIME is set use start_adapter launcher to run adapter
- if "%WBIA_RUNTIME%"=="" goto CROSSWORLDS
- REM Call CWConnEnv
  call "%WBIA_RUNTIME%"\bin\VWConnEnv
- REM set the directory where the specific connector resides
  set CONNDIR="%WBIA_RUNTIME%"\connectors\%1
- REM goto the connector specific drive & directory
  cd/d%CONNDIR%
- REM Set variables needed for start_adapter callee
- set JVMArgs=
  set JCLASSES=.;%CW_I2JAR%;%I2PROPERTIES%;%JCLASSES%
  set LibPath=
  set ExtDirs
- call "%WBIA_RUNTIME%\bin\start_adapter" -n%CONNAME% -s%SERVER% -1%CONNPACKAGENAME%%3%4%5

- pause
  goto END
  :CROSSWORLDS
- REM call CWConnEnv
  call "%CROSSWORLDS%"\bin\CWConnEnv
- REM set the directory where the specific connector resides
  set CONNDIR="%CROSSWORLDS%"\connectors\%1
- REM goto the connector specific drive & directory
  cd/d%CONNDIR%
- set JCLASSES=.;%CW_I2JAR%;%I2JAR;%I2PROPERTIES%;%JCLASSES%
- REM config file location defaults to HOME\InterchangeSystem.cfg on the local machine
- REM start the java connector under the Java Application End
- %CWJAVA% -mx128m %ORB_PROPERTY%
  -Djava.ext.dirs="%MQ_LIB%";%JRE_EXT_DIRS%
  -Djava.library.path="%CROSSWORLDS%"\bin;%CONNDIR%;
  "%MQ_LIB%;%JRE_EXT_DIRS%
  -Duser.home="%CROSSWORLDS%" -classpath %JCLASSES% AppEndWrapper
  -1%CONNPACKAGENAME% -n%CONNAME% -s%SERVER% %3 %4 %5
- :END
  endlocal
- Pause

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.
  - When using InterChange Server 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:

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 29.

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.

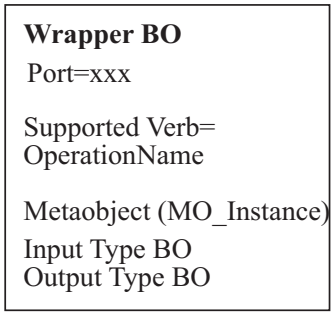
The business objects are generated using the WebSphere Business Integration Adapter utility XML ODA, which reads the XML schema files for these types and generates the corresponding IBM business object. (See Chapter 4, “Generating business objects using the i2 ODA,” on page 29 and Chapter 3, "XML data 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 29.

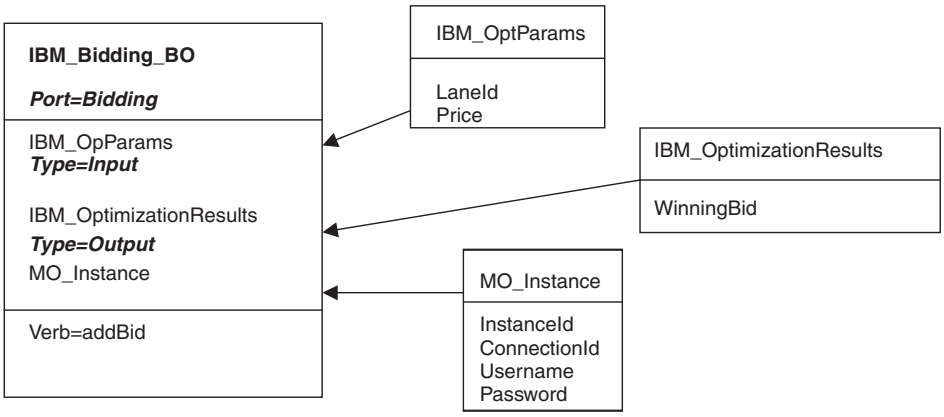
The following diagram shows the parts of a wrapper business object. A description of each part follows the diagram.



- 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 meta-objects for polling" on page 22.
- 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 meta-objects 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.

All of the attributes within the meat-object require a static default value. The attributes are within the metaobjects have a static default value.

- instanceId- The i2 CIS instance associated with the polling subscription.
- connectionId- ID used for the persistent connection (during event retrieval).
- username- The user used to create the connection for event retrival.
- password- The user password.
- OperationName- The operation to execute when retrieving the event from the i2 CIS.
- WrapperBOName- The name of the wrapper business object which provides containment for the event business object for the operation.
- Unregister- The boolean value used to determine whether the adapter should unregister its durable subscription when terminating.
- AutoAcknowledge- The boolean value used to determine whether the adapter should enable redelivery of failed events.

The polling meta-bject has the following business object level application specific information:

- Port- Used to detemine which i2 CIS port or i2 CIS adapter this polling subscription should be registered.

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.

Unregistering polling subscriptions

Polling subscriptions can be unregistered by altering the polling meta-object property. A new attribute `unregister` that accepts Boolean values is used to determine whether the adapter should unregister for this operation when the adapter is shut down. Connections created with the `unregister` operations value set to `false` will not be destroyed.

Event redelivery

Events can be redelivered when the AutoAcknowledge property is set to false. This allows for the adapter to enable the i2 message bus to retain the event until it is successfully sent to the integration broker. This, in conjunction with enabling the adapter to keep its registration for events, will provide guaranteed message delivery.

Note: If AutoAcknowledge is set to false, the connectionId attribute is mandatory and must not be reused by any other polling meta-object. The default behavior is AutoAcknowledge true.

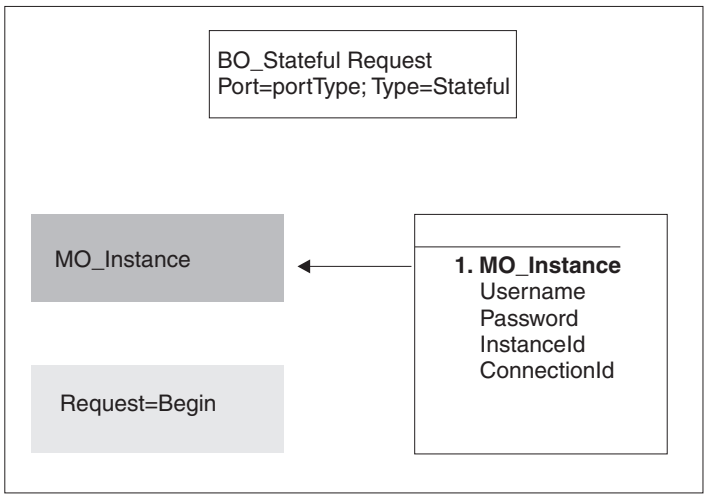
Using metaobjects for stateful operations

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.

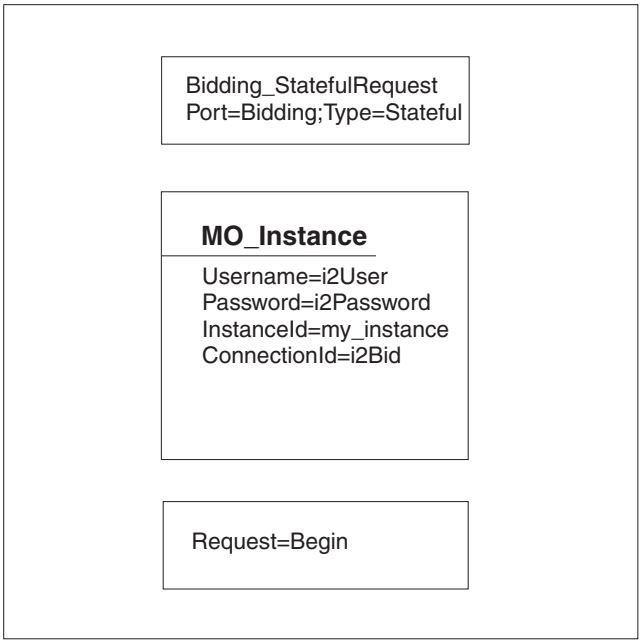
The following examples illustrate using metaobjects for stateful operations.

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″.

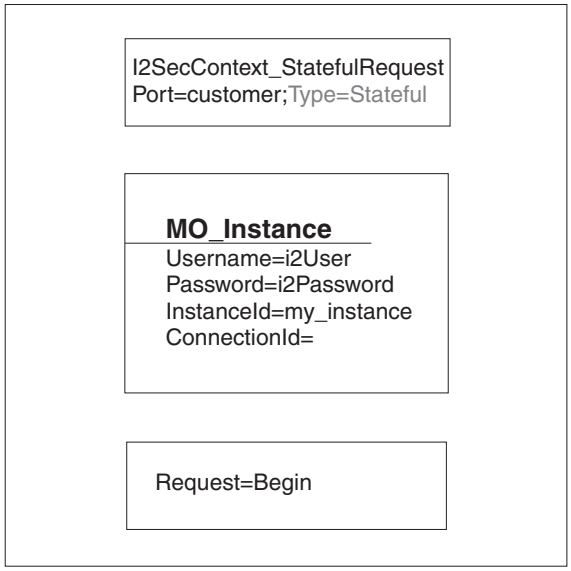
Note: The connectionId is what associates a series of stateful operations with each other; therefore, for each series of bids a different connectionId should be provided. The broker can also allow the i2 CIS agent to generate the connectionId, which will be returned on the ″Request=Begin″ request call 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.
IsForeighKey	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
IsForeighKey	Not used.
Is Required	For relationship details between XML elements and requiredness, see Chapter 3, "XML data handler," in the *Data Handler Guide*.
AppSpecInfo	For information on this property, see "Identifying business object application-specific information" on page 27,
Cardinality	For relationship details between XML elements and cardinality, see Chapter 3, "XML data handler," in the *Data Handler Guide*.

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 `CxBlank`. Simple (String) attributes with a value of `CxBlank` are included in an XML document. Empty double quotation marks (`""`) in an XML document are used as the PCDATA equivalent of `CxBlank`.

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

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 31.

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 9). 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. 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 CONNECTORJAR=
```

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

You 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 `i2ODAAgent.txt`) are located in `\ODA\messages\`, which is under the product directory. These files use the following naming convention:

AgentName`Agent.txt`

Example: If the *AGENTNAME* variable specifies *i2ODA1*, the tool assumes that the name of the associated message file is `i2ODA1Agent.txt`

You can have a message file for each ODA instance or have differently named ODAs use the same message file.

The message file name and the tracing levels are configured by default when you install the adapter.

The following table describes the tracing levels.

Trace Level	Description
0	• 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	• 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 32.

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.

Note: You can also find the agent using the host name and port number

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.

   You can run multimple instances of the i2 ODA on either the local host or the remote host in the network. Each instance runs on a unique port. 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. **Example:** i2_BO
2	SchemaFileLocation	String	Path where the generated.`xsd` files are stored. This is mandatory; you must specify the path to store the schema files.
3	CISAgentHostName	String	Host name of the CIS agent, it is specified if the CIS agent is running on a remote machine.
4	MetadataService	String	Name of the i2 CIS port where the MetadataService is located. The default is `MetadataService`.
5	PortTypesOperation	String	Name of the operation to get the port types through metadata introspection. The default is `getPorttypes`.
6	SchemasOperation	String	Name of the operation to get the schemas for the selected operations. The default is `getSchemasForOperation`.

The following parameters are configured by default when you install the adapter, so you are not prompted to configure them during the initial communication between Business Object Designer and the i2 ODA.

- MessageFile--Path to the error and message file. The i2 ODA displays the file name according to the naming convention.

  Example: If the agent is named `i2ODA`, the value of the message file property displays as `i2ODAAgent.txt`.
- 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 `i2ODA`, it generates a trace file named `i2ODAtrace.txt`.
- TraceLevel--Level of tracing enabled for the i2 ODA. For more information, see "Working with error and trace message files" on page 30.

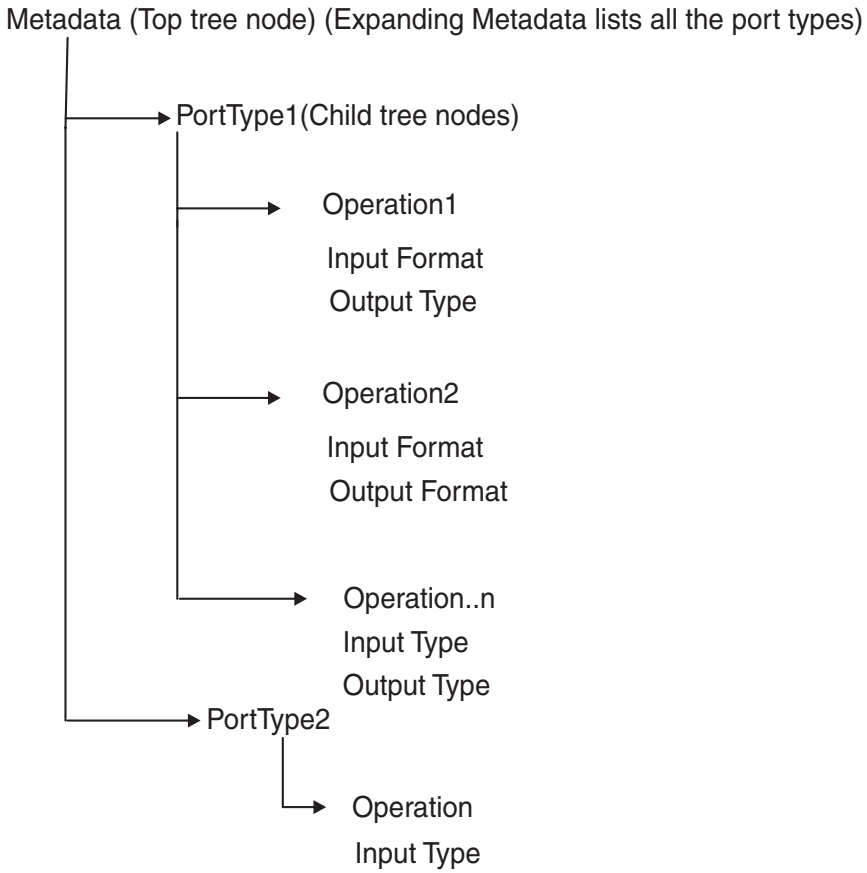
Expand nodes and select port types, operations, and input/output types/formats

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.

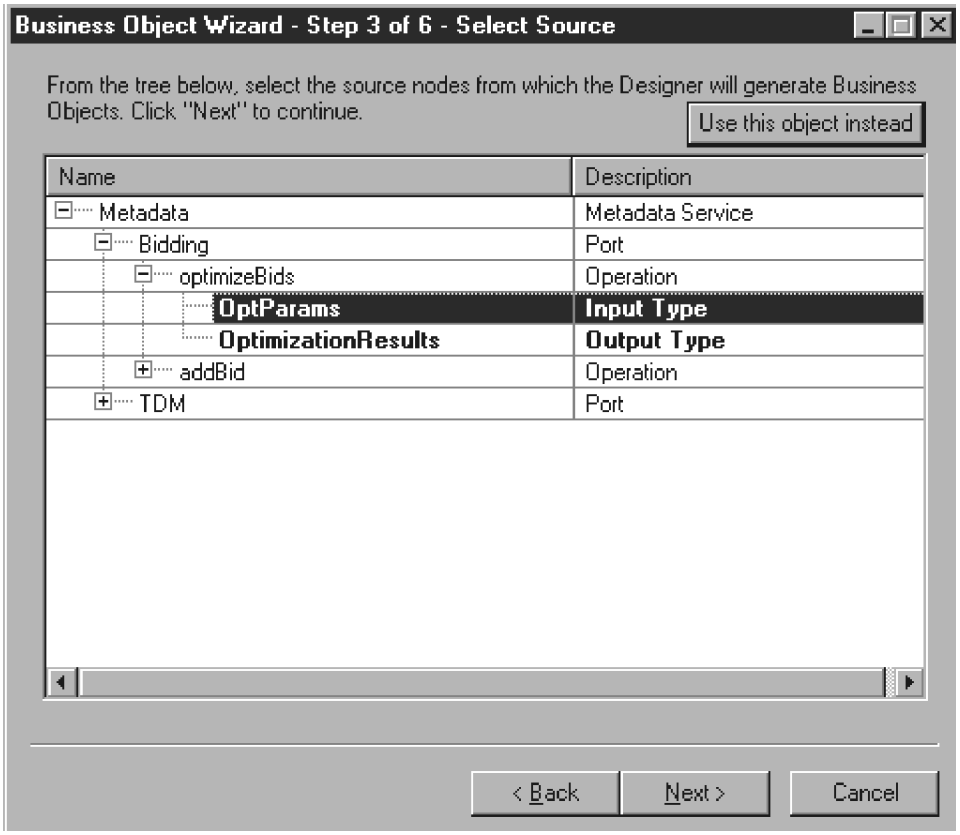
A distinction is made between ″Types″ and ″Formats″ when dealing with the input and output values for an operation. When displaying the input and output business objects for an operation, if it is a ″Format″ that will be displayed instead of a ″Type,″ it will display as ″Output Format″ instead of ″Output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

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

The following diagram shows a tree view with expanded nodes.



- 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.

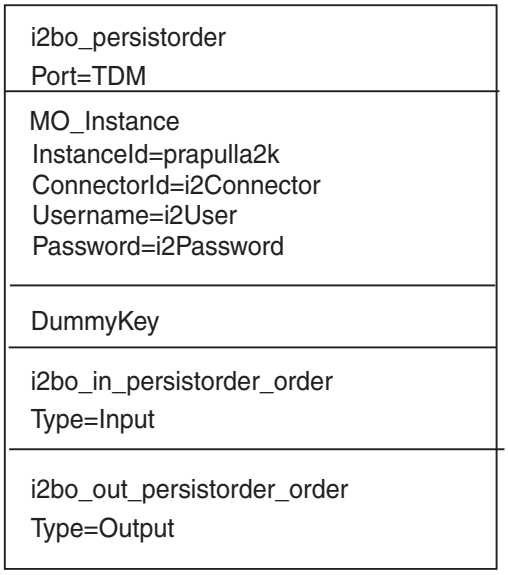
   Result: The i2 ODA creates a wrapper business object to represent the operation with the chosen port type. Each operation has one wrapper business object. The input and output types of the operation form the attributes of this wrapper business object, with the operation as the verb. The port type becomes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

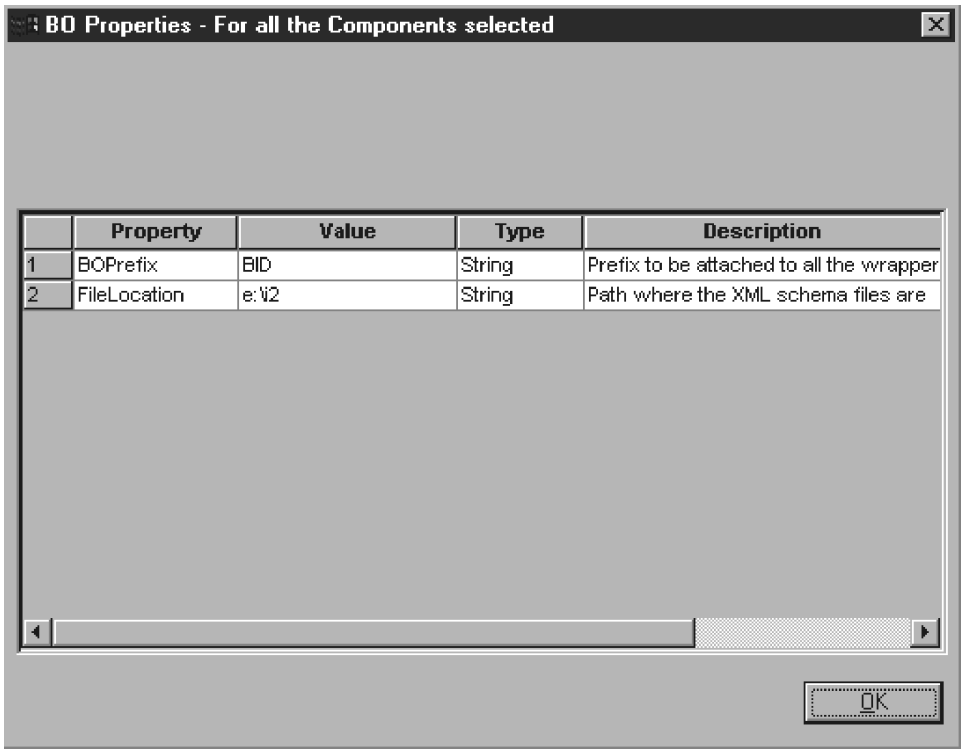


2. Save the wrapper business object to a file. Saving it to the InterChange Server will fail because the corresponding dependent attribute business objects have not yet been created by the XML ODA schema.

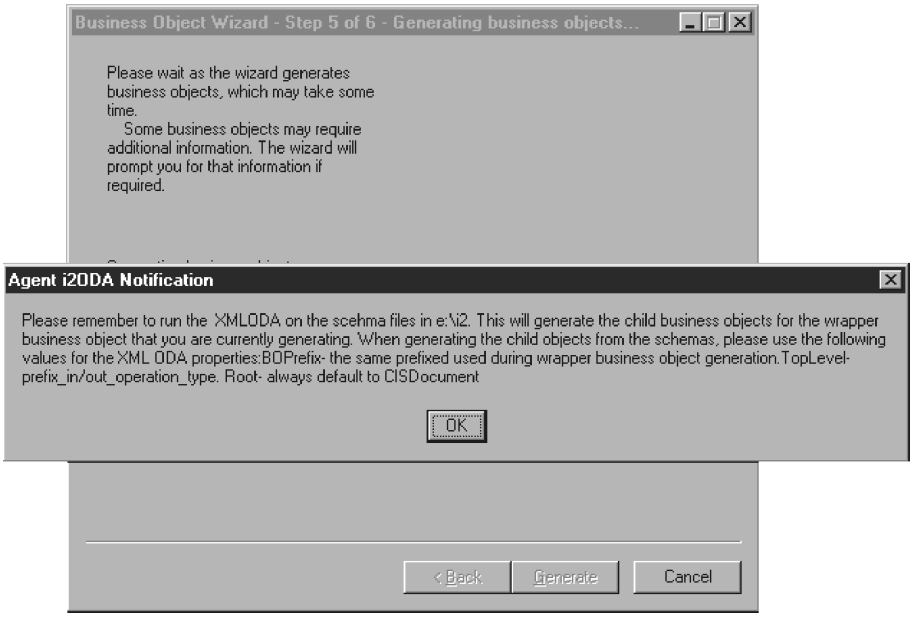
   Guideline: When you save the business objects into a file in this step, the current Business Object Designer wizard looks at the machine where the agent is running. Another pop-up window prompts you for a location to save the generated wrapper business objects.

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.

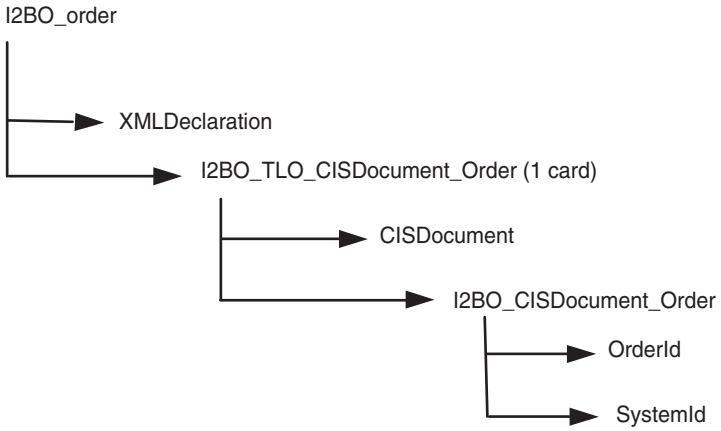
      Guideline: Remember to run the XML ODA on the schema files in the directory you specified when running the ODA. This will generate the child

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

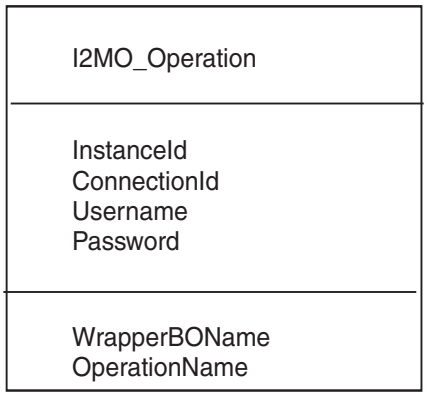
Property	Value
Doctypeor SchemaLocation	true
TraceFileName	XMLODAtrace.txt
TraceLevel	5
MessageFile	XMLODAAgent.txt

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.

Creating 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 19.

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

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 19. 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	• Messages logged each time a business object is posted to the integration broker from gotApplEvent. • Messages that indicate each time a business object request is received.
Level 3	Not applicable
Level 4	• 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	• 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 -Djava.rmi.server.hostname=<hostname> -m
    <metadatabinding file>
   ```

4. Start the CIS agent giving the same JVM argument:

   ```
   -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 with 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 (and shown as WMQI in the Connector Configurator).
- Information Integrator (II)
- WebSphere Application Server (WAS)

If your adapter supports DB2 Information Integrator, use the WMQI options and the DB2 II standard properties (see the Notes column in Table 2 on page 49.)

The properties you set for the adapter depend on which integration broker you use. You choose the integration broker using Connector Configurator. After you choose the broker, Connector Configurator lists the standard properties you must configure for the adapter.

For information about properties specific to this connector, see the relevant section in this guide.

New properties

This standard property was added in this release:

- BOTrace

Standard connector properties overview

Connectors have two types of configuration properties:

- Standard configuration properties, which are used by the framework
- Application, or connector-specific, configuration properties, which are used by the agent

These properties determine the adapter framework and the agent run-time behavior.

This section describes how to start Connector Configurator and describes characteristics common to all properties. For information on configuration properties specific to a connector, see its adapter user guide.

Starting Connector Configurator

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

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.

Configuration property values overview

The connector uses the following order to determine a property's value:

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

The default length of a property field is 255 characters. There is no limit on the length of a STRING property type. The length of an INTEGER type is determined by the server on which the adapter is running.

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.

The update characteristics of a property, that is, how and when a change to the connector properties takes effect, depend on the nature of the property.

There are four update methods for standard connector properties:

- **Dynamic**
  The new value takes effect immediately after the change is saved in System Manager. However, if the connector is in stand-alone mode (independently of System Manager), for example, if it is running with one of the WebSphere message brokers, you can change properties only through the configuration file. In this case, a dynamic update is not possible.
- **Agent restart (ICS only)**
  The new value takes effect only after you stop and restart the connector agent.
- **Component restart**
  The new value takes effect only after the connector is stopped and then restarted in System Manager. You do not need to stop and restart the agent or the server process.
- **System restart**
  The new value takes effect only after you stop and restart the connector agent and the server.

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 49.

There are three locations in which a standard property can reside. Some properties can reside in more than one location.

- **ReposController**
  The property resides in the connector controller and is effective only there. If you change the value on the agent side, it does not affect the controller.
- **ReposAgent**
  The property resides in the agent and is effective only there. A local configuration can override this value, depending on the property.

- **LocalConfig**
  The property resides in the configuration file for the connector and can act only through the configuration file. The controller cannot change the value of the property, and is not aware of changes made to the configuration file unless the system is redeployed to update the controller explicitly.

Standard properties quick-reference

Table 2 provides a quick-reference to the standard connector configuration properties. Not all connectors require all of these properties, and property settings may differ from integration broker to integration broker.

See the section following the table for a description of each property.

Note: In the Notes column in Table 2, the phrase "RepositoryDirectory is set to <REMOTE>" indicates that the broker is InterChange Server. When the broker is WMQI or WAS, the repository directory is set to *<ProductDir>*\repository

Table 2. Summary of standard configuration properties

Property name	Possible values	Default value	Update method	Notes
AdapterHelpName	One of the valid subdirectories in *<ProductDir>*\bin\Data \App\Help\ that contains a valid *<RegionalSetting>* directory	Template name, if valid, or blank field	Component restart	Supported regional settings. Include `chs_chn`, `cht_twn`, `deu_deu`, `esn_esp`, `fra_fra`, `ita_ita`, `jpn_jpn`, `kor_kor`, `ptb_bra`, and `enu_usa` (default).
AdminInQueue	Valid JMS queue name	*<CONNECTORNAME>* `/ADMININQUEUE`	Component restart	This property is valid only when the value of DeliveryTransport is `JMS`
AdminOutQueue	Valid JMS queue name	*<CONNECTORNAME>* `/ADMINOUTQUEUE`	Component restart	This property is valid only when the value of DeliveryTransport is `JMS`
AgentConnections	`1` through `4`	`1`	Component restart	This property is valid only when the value of DeliveryTransport is `MQ` or `IDL`, the value of Repository Directory is set to <REMOTE> and the value of BrokerType is ICS.
AgentTraceLevel	`0` through `5`	`0`	Dynamic if broker is ICS; otherwise Component restart	
ApplicationName	Application name	The value specified for the connector application name	Component restart	

Table 2. Summary of standard configuration properties (continued)

Property name	Possible values	Default value	Update method	Notes
BiDi.Application	Any valid combination of these bidirectional attributes: 1st letter: I,V 2nd letter: L,R 3rd letter: Y, N 4th letter: S, N 5th letter: H, C, N	ILYNN (five letters)	Component restart	This property is valid only if the value of BiDi.Transforma tion is true
BiDi.Broker	Any valid combination of these bidirectional attributes: 1st letter: I,V 2nd letter: L,R 3rd letter: Y, N 4th letter: S, N 5th letter: H, C, N	ILYNN (five letters)	Component restart	This property is valid only if the value of BiDi.Transformation is true. If the value of BrokerType is ICS, the property is read-only.
BiDi.Metadata	Any valid combination of these bidirectional attributes: 1st letter: I,V 2nd letter: L,R 3rd letter: Y, N 4th letter: S, N 5th letter: H, C, N	ILYNN (five letters)	Component restart	This property is valid only if the value of BiDi.Transformation is true.
BiDi.Transformation	true or false	false	Component restart	This property is valid only if the value of BrokerType is not WAS.
BOTrace	none or keys or full	none	Agent restart	This property is valid only if the value of AgentTraceLevel is lower than 5.
BrokerType	ICS, WMQI, WAS	ICS	Component restart	
CharacterEncoding	Any supported code. The list shows this subset: ascii7, ascii8, SJIS, Cp949, GBK, Big5, Cp297, Cp273, Cp280, Cp284, Cp037, Cp437 .	ascii7	Component restart	This property is valid only for C++ connectors.
CommonEventInfrastruc ture	true or false	false	Component restart	
CommonEventInfrastruc tureURL	A URL string, for example, corbaloc:iiop: host:2809.	No default value.	Component restart	This property is valid only if the value of CommonEvent Infrastructure is true.
ConcurrentEventTrig geredFlows	1 through 32,767	1	Component restart	This property is valid only if the value of RepositoryDirectory is set to <REMOTE> and the value of BrokerType is ICS.
ContainerManagedEvents	Blank or JMS	Blank	Component restart	This property is valid only when the value of Delivery Transport is JMS.

Table 2. Summary of standard configuration properties (continued)

Property name	Possible values	Default value	Update method	Notes
ControllerEventSequencing	true or false	true	Dynamic	This property is valid only if the value of Repository Directory is set to <REMOTE> and the value of BrokerType is ICS.
ControllerStoreAndForwardMode	true or false	true	Dynamic	This property is valid only if the value of Repository Directory is set to <REMOTE> and the value of BrokerType is ICS.
ControllerTraceLevel	0 through 5	0	Dynamic	This property is valid only if the value of RepositoryDirectory is set to <REMOTE> and the value of BrokerType is ICS.
DeliveryQueue	Any valid JMS queue name	*<CONNECTORNAME>*/DELIVERYQUEUE	Component restart	This property is valid only when the value of Delivery Transport is JMS.
DeliveryTransport	MQ, IDL, or JMS	IDL when the value of RepositoryDirectory is <REMOTE>, otherwise JMS	Component restart	If the value of RepositoryDirectory is not <REMOTE>, the only valid value for this property is JMS.
DuplicateEventElimination	true or false	false	Component restart	This property is valid only if the value of DeliveryTransport is JMS.
EnableOidForFlowMonitoring	true or false	false	Component restart	This property is valid only if the value of BrokerType is ICS.
FaultQueue	Any valid queue name.	*<CONNECTORNAME>*/FAULTQUEUE	Component restart	This property is valid only if the value of DeliveryTransport is JMS.
jms.FactoryClassName	CxCommon.Messaging.jms.IBMMQSeriesFactory, CxCommon.Messaging.jms.SonicMQFactory, or any Java class name	CxCommon.Messaging.jms.IBMMQSeriesFactory	Component restart	This property is valid only if the value of DeliveryTransport is JMS.
jms.ListenerConcurrency	1 through 32767	1	Component restart	This property is valid only if the value of jms.TransportOptimized is true.
jms.MessageBrokerName	If the value of jms.FactoryClassName is IBM, use crossworlds.queue.manager.	crossworlds.queue.manager	Component restart	This property is valid only if the value of DeliveryTransport is JMS .
jms.NumConcurrentRequests	Positive integer	10	Component restart	This property is valid only if the value of DeliveryTransport is JMS .

Table 2. Summary of standard configuration properties (continued)

Property name	Possible values	Default value	Update method	Notes
jms.Password	Any valid password		Component restart	This property is valid only if the value of DeliveryTransport is JMS .
jms.TransportOptimized	true or false	false	Component restart	This property is valid only if the value of DeliveryTransport is JMS and the value of BrokerType is ICS.
jms.UserName	Any valid name		Component restart	This property is valid only if the value of Delivery Transport is JMS.
JvmMaxHeapSize	Heap size in megabytes	128m	Component restart	This property is valid only if the value of Repository Directory is set to <REMOTE> and the value of BrokerType is ICS.
JvmMaxNativeStackSize	Size of stack in kilobytes	128k	Component restart	This property is valid only if the value of Repository Directory is set to <REMOTE> and the value of BrokerType is ICS.
JvmMinHeapSize	Heap size in megabytes	1m	Component restart	This property is valid only if the value of Repository Directory is set to <REMOTE> and the value of BrokerType is ICS.
ListenerConcurrency	1 through 100	1	Component restart	This property is valid only if the value of DeliveryTransport is MQ.
Locale	This is a subset of the supported locales: en_US, ja_JP, ko_KR, zh_CN, zh_TW, fr_FR, de_DE, it_IT, es_ES, pt_BR	en_US	Component restart	
LogAtInterchangeEnd	true or false	false	Component restart	This property is valid only if the value of Repository Directory is set to <REMOTE> and the value of BrokerType is ICS.
MaxEventCapacity	1 through 2147483647	2147483647	Dynamic	This property is valid only if the value of Repository Directory is set to <REMOTE> and the value of BrokerType is ICS.
MessageFileName	Valid file name	InterchangeSystem.txt	Component restart	

Table 2. Summary of standard configuration properties (continued)

Property name	Possible values	Default value	Update method	Notes
MonitorQueue	Any valid queue name	<*CONNECTORNAME*> /MONITORQUEUE	Component restart	This property is valid only if the value of DuplicateEventElimination is `true` and ContainerManagedEvents has no value.
OADAutoRestartAgent	`true` or `false`	`false`	Dynamic	This property is valid only if the value of Repository Directory is set to <REMOTE> and the value of BrokerType is ICS.
OADMaxNumRetry	A positive integer	`1000`	Dynamic	This property is valid only if the value of Repository Directory is set to <REMOTE> and the value of BrokerType is ICS.
OADRetryTimeInterval	A positive integer in minutes	`10`	Dynamic	This property is valid only if the value of Repository Directory is set to <REMOTE> and the value of BrokerType is ICS.
PollEndTime	HH = 0 through 23 MM = 0 through 59	`HH:MM`	Component restart	
PollFrequency	A positive integer (in milliseconds)	`10000`	Dynamic if broker is ICS; otherwise Component restart	
PollQuantity	1 through `500`	`1`	Agent restart	This property is valid only if the value of ContainerManagedEvents is `JMS`.
PollStartTime	HH = 0 through 23 MM = 0 through 59	`HH:MM`	Component restart	
RepositoryDirectory	<REMOTE> if the broker is ICS; otherwise any valid local directory.	For ICS, the value is set to <REMOTE> For WMQI and WAS, the value is <*ProductDir* `\repository`	Agent restart	
RequestQueue	Valid JMS queue name	<*CONNECTORNAME*> /REQUESTQUEUE	Component restart	This property is valid only if the value of DeliveryTransport is `JMS`
ResponseQueue	Valid JMS queue name	<*CONNECTORNAME*> /RESPONSEQUEUE	Component restart	This property is valid only if the value of DeliveryTransport is `JMS`.
RestartRetryCount	`0` through `99`	`7`	Dynamic if ICS; otherwise Component restart	

Table 2. Summary of standard configuration properties (continued)

Property name	Possible values	Default value	Update method	Notes
RestartRetryInterval	A value in minutes from 1 through 2147483647	1	Dynamic if ICS; otherwise Component restart	
ResultsSetEnabled	true or false	false	Component restart	Used only by connectors that support DB2II. This property is valid only if the value of DeliveryTransport is JMS, and the value of BrokerType is WMQI.
ResultsSetSize	Positive integer	0 (means the results set size is unlimited)	Component restart	Used only by connectors that support DB2II. This property is valid only if the value of ResultsSetEnabled is true.
RHF2MessageDomain	mrm or xml	mrm	Component restart	This property is valid only if the value of DeliveryTransport is JMS and the value of WireFormat is CwXML.
SourceQueue	Any valid WebSphere MQ queue name	*<CONNECTORNAME>*/SOURCEQUEUE	Agent restart	This property is valid only if the value of ContainerManagedEvents is JMS.
SynchronousRequest Queue	Any valid queue name.	*<CONNECTORNAME>*/SYNCHRONOUSREQUEST QUEUE	Component restart	This property is valid only if the value of DeliveryTransport is JMS.
SynchronousRequest Timeout	0 to any number (milliseconds)	0	Component restart	This property is valid only if the value of DeliveryTransport is JMS.
SynchronousResponse Queue	Any valid queue name	*<CONNECTORNAME>*/SYNCHRONOUSRESPONSE QUEUE	Component restart	This property is valid only if the value of DeliveryTransport is JMS.
TivoliMonitorTransaction Performance	true or false	false	Component restart	
WireFormat	CwXML or CwBO	CwXML	Agent restart	The value of this property must be CwXML if the value of RepositoryDirectory is not set to <REMOTE>. The value must be CwBO if the value of RepositoryDirectory is set to <REMOTE>.
WsifSynchronousRequest Timeout	0 to any number (milliseconds)	0	Component restart	This property is valid only if the value of BrokerType is WAS.

Table 2. Summary of standard configuration properties (continued)

Property name	Possible values	Default value	Update method	Notes
XMLNameSpaceFormat	short or long or no	short	Agent restart	This property is valid only if the value of BrokerType is WMQI or WAS

Standard properties

This section describes the standard connector configuration properties.

AdapterHelpName

The AdapterHelpName property is the name of a directory in which connector-specific extended help files are located. The directory must be located in <*ProductDir*>\bin\Data\App\Help and must contain at least the language directory enu_usa. It may contain other directories according to locale.

The default value is the template name if it is valid, or it is blank.

AdminInQueue

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

The default value is <*CONNECTORNAME*>/ADMININQUEUE

AdminOutQueue

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

The default value is <*CONNECTORNAME*>/ADMINOUTQUEUE

AgentConnections

The AgentConnections property controls the number of ORB (Object Request Broker) connections opened when the ORB initializes.

It is valid only if the value of the RepositoryDirectory is set to <REMOTE> and the value of the DeliveryTransport property is MQ or IDL.

The default value of this property is 1.

AgentTraceLevel

The AgentTraceLevel property sets the level of trace messages for the application-specific component. The connector delivers all trace messages applicable at the tracing level set and lower.

The default value is 0.

ApplicationName

The ApplicationName property uniquely identifies the name of the connector application. This name is used by the system administrator to monitor the integration environment. This property must have a value before you can run the connector.

The default is the name of the connector.

BiDi.Application

The BiDi.Application property specifies the bidirectional format for data coming from an external application into the adapter in the form of any business object supported by this adapter. The property defines the bidirectional attributes of the application data. These attributes are:

- Type of text: implicit or visual (`I` or `V`)
- Text direction: left-to-right or right-to-left (`L` or `R`)
- Symmetric swapping: on or off (`Y` or `N`)
- Shaping (Arabic): on or off (`S` or `N`)
- Numerical shaping (Arabic): Hindi, contextual, or nominal (`H`, `C`, or `N`)

This property is valid only if the BiDi.Transformation property value is set to true.

The default value is `ILYNN` (implicit, left-to-right, on, off, nominal).

BiDi.Broker

The BiDi.Broker property specifies the bidirectional script format for data sent from the adapter to the integration broker in the form of any supported business object. It defines the bidirectional attributes of the data, which are as listed under `BiDi.Application` above.

This property is valid only if the BiDi.Transformation property value is set to true. If the BrokerType property is ICS, the property value is read-only.

The default value is `ILYNN` (implicit, left-to-right, on, off, nominal).

BiDi.Metadata

The BiDi.Metadata property defines the bidirectional format or attributes for the metadata, which is used by the connector to establish and maintain a link to the external application. The attribute settings are specific to each adapter using the bidirectional capabilities. If your adapter supports bidirectional processing, refer to the section on adapter-specific properties for more information.

This property is valid only if the BiDi.Transformation property value is set to true.

The default value is `ILYNN` (implicit, left-to-right, on, off, nominal).

BiDi.Transformation

The BiDi.Transformation property defines whether or not the system performs a bidirectional transformation at run time.

If the property value is set to `true`, the BiDi.Application, BiDi.Broker, and BiDi.Metadata properties are available. If the property value is set to `false`, they are hidden.

The default value is `false`.

BOTrace

The BOTrace property specifies whether or not business object trace messages are enabled at run time.

Note: It applies only when the AgentTraceLevel property is set to less than 5.

When the trace level is set to less than 5, you can use these command line parameters to reset the value of BOTrace.

- Enter `-xBOTrace=Full` to dump all the business object's attributes.
- Enter `-xBOTrace=Keys` to dump only the business object's keys.
- Enter `-xBOTrace=None` to disable business object attribute dumping.

The default value is `false`.

BrokerType

The BrokerType property identifies the integration broker type that you are using. The possible values are `ICS`, `WMQI (for WMQI, WMQIB or WBIMB)`, or `WAS`.

CharacterEncoding

The CharacterEncoding property 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. C++ connectors use the value `ascii7` for this property.

By default, only a subset of supported character encodings is displayed. To add other supported values to the list, you must manually modify the \Data\Std\stdConnProps.xml file in the product directory (<*ProductDir*>). For more information, see the Connector Configurator appendix in this guide.

CommonEventInfrastructure

The Common Event Infrastructure (CEI) is a simple event management function handling generated events. The CommonEventInfrastructure property specifies whether the CEI should be invoked at run time.

The default value is `false`.

CommonEventInfrastructureContextURL

The CommonEventInfrastructureContextURL is used to gain access to the WAS server that executes the Common Event Infrastructure (CEI) server application. This property specifies the URL to be used.

This property is valid only if the value of CommonEventInfrastructure is set to `true.`

The default value is a blank field.

ConcurrentEventTriggeredFlows

The ConcurrentEventTriggeredFlows property determines how many business objects can be concurrently processed by the connector for event delivery. You set the value of this attribute to the number of business objects that are mapped and delivered concurrently. For example, if you set the value of this property to 5, five business objects are processed concurrently.

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), the following properties must configured:

- The collaboration must be configured to use multiple threads by setting its `Maximum number of concurrent events` property high enough to use multiple threads.
- The destination application's application-specific component must be configured to process requests concurrently. That is, it must be multithreaded, or it must be able to use connector agent parallelism and be configured for multiple processes. The Parallel Process Degree configuration property must be set to a value larger than 1.

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

This property is valid only if the value of the RepositoryDirectory property is set to <REMOTE>.

The default value is 1.

ContainerManagedEvents

The ContainerManagedEvents 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 one JMS transaction.

When this property is set to `JMS`, the following properties must also be set to enable guaranteed event delivery:

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

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

Although these properties are adapter-specific, here are some example values:

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

The fields for these values in the **Data Handler** tab are displayed only if you have set the ContainerManagedEvents property to the value `JMS`.

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

The ContainerManagedEvents property is valid only if the value of the DeliveryTransport property is set to `JMS`.

There is no default value.

ControllerEventSequencing

The ControllerEventSequencing property enables event sequencing in the connector controller.

This property is valid only if the value of the RepositoryDirectory property is set to set to <REMOTE> (`BrokerType` is `ICS`).

The default value is `true`.

ControllerStoreAndForwardMode

The ControllerStoreAndForwardMode property 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.

This property is valid only if the value of the RepositoryDirectory property is set to <REMOTE> (the value of the BrokerType property is ICS).

The default value is `true`.

ControllerTraceLevel

The ControllerTraceLevel property sets the level of trace messages for the connector controller.

This property is valid only if the value of the RepositoryDirectory property is set to set to <REMOTE>.

The default value is `0`.

DeliveryQueue

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

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

The default value is <*CONNECTORNAME*>/DELIVERYQUEUE.

DeliveryTransport

The DeliveryTransport property 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 value of the RepositoryDirectory property is set to <REMOTE>, the value of the DeliveryTransport property can be MQ, IDL, or JMS, and the default is IDL.
- If the value of the RepositoryDirectory property is a local directory, the value can be only JMS.

The connector sends service-call requests and administrative messages over CORBA IIOP if the value of the RepositoryDirectory property is MQ or IDL.

If the value of the DeliveryTransport property is MQ, you can set the command-line parameter WhenServerAbsent in the adapter start script to indicate whether the adapter should pause or shut down when the InterChange Server is shut down.

- Enter WhenServerAbsent=pause to pause the adapter when ICS is not available.
- Enter WhenServerAbsent=shutdown to shut down the adapter when ICS is not available.

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 prevents writing 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 polling thread picks up an event, places it in the connector queue, then picks up the next event. This is faster than IDL, which requires the connector polling thread to pick up an event, go across the network into the server process, store the event persistently in the repository database, then pick up the next event.

JMS

The JMS transport mechanism 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

are listed in Connector Configurator. The properties `jms.MessageBrokerName` and `jms.FactoryClassName` are required for this transport.

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
- 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 768MB of process heap size, set the following variable and property:

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

  This script is located in the \bin directory below the product directory (<*ProductDir*>). Using 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 larger than this limit, page swapping can occur, which can adversely affect the performance of your system.
- Set the value of the IPCCBaseAddress property to 11 or 12. For more information on this property, see the *System Installation Guide for UNIX*.

DuplicateEventElimination

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

Note: When the value of this property is `true`, the MonitorQueue property must be enabled to provide guaranteed event delivery.

The default value is `false`.

EnableOidForFlowMonitoring

When the value of this property is `true`, the adapter runtime will mark the incoming ObjectEventID as a foreign key for flow monitoring.

This property is only valid if the BrokerType property is set to ICS.

The default value is `false`.

FaultQueue

If the connector experiences an error while processing a message, it moves the message (and a status indicator and description of the problem) to the queue specified in the FaultQueue property.

The default value is *<CONNECTORNAME>*`/FAULTQUEUE`.

jms.FactoryClassName

The jms.FactoryClassName property specifies the class name to instantiate for a JMS provider. This property must be set if the value of the DeliveryTransport property is `JMS`.

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

jms.ListenerConcurrency

The jms.ListenerConcurrency property specifies the number of concurrent listeners for the JMS controller. It specifies the number of threads that fetch and process messages concurrently within a controller.

This property is valid only if the value of the jms.OptimizedTransport property is `true`.

The default value is `1`.

jms.MessageBrokerName

The jms.MessageBrokerName specifies the broker name to use for the JMS provider. You must set this connector property if you specify `JMS` as the delivery transport mechanism (in the DeliveryTransport property).

When you connect to a remote message broker, this property requires the following values:
`QueueMgrName``:`*`Channel`*`:`*`HostName`*`:`*`PortNumber`*
where:
`QueueMgrName` is the name of the queue manager.
`Channel` is the channel used by the client.
`HostName` is the name of the machine where the queue manager is to reside.
*`PortNumber`*is the port number used by the queue manager for listening

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

The default value is `crossworlds.queue.manager`. Use the default when connecting to a local message broker.

jms.NumConcurrentRequests

The jms.NumConcurrentRequests property 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 are blocked and must wait for another request to complete before proceeding.

The default value is `10`.

jms.Password

The jms.Password property specifies the password for the JMS provider. A value for this property is optional.

There is no default value.

jms.TransportOptimized

The jms.TransportOptimized property determines if the WIP (work in progress) is optimized. You must have a WebSphere MQ provider to optimize the WIP. For optimized WIP to operate, the messaging provider must be able to:

1. Read a message without taking it off the queue
2. Delete a message with a specific ID without transferring the entire message to the receiver's memory space
3. Read a message by using a specific ID (needed for recovery purposes)
4. Track the point at which events that have not been read appear.

The JMS APIs cannot be used for optimized WIP because they do not meet conditions 2 and 4 above, but the MQ Java APIs meet all four conditions, and hence are required for optimized WIP.

This property is valid only if the value of DeliveryTransport is `JMS` and the value of BrokerType is `ICS`.

The default value is `false`.

jms.UserName

the jms.UserName property specifies the user name for the JMS provider. A value for this property is optional.

There is no default value.

JvmMaxHeapSize

The JvmMaxHeapSize property specifies the maximum heap size for the agent (in megabytes).

This property is valid only if the value for the RepositoryDirectory property is set to <REMOTE>.

The default value is 128m.

JvmMaxNativeStackSize

The JvmMaxNativeStackSize property specifies the maximum native stack size for the agent (in kilobytes).

This property is valid only if the value for the RepositoryDirectory property is set to <REMOTE>.

The default value is 128k.

JvmMinHeapSize

The JvmMinHeapSize property specifies the minimum heap size for the agent (in megabytes).

This property is valid only if the value for the RepositoryDirectory property is set to <REMOTE>.

The default value is 1m.

ListenerConcurrency

The ListenerConcurrency property supports multithreading in WebSphere MQ Listener when ICS is the integration broker. It enables batch writing of multiple events to the database, thereby improving system performance.

This property valid only with connectors that use MQ transport. The value of the DeliveryTransport property must be `MQ`.

The default value is 1.

Locale

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

A locale name has the following format:

ll_TT.codeset

where:
ll is a two-character language code (in lowercase letters)
TT is a two-letter country or territory code (in uppercase letters)
codeset is the name of the associated character code set (may be optional).

By default, only a subset of supported locales are listed. To add other supported values to the list, you modify the \Data\Std\stdConnProps.xml file in the <*ProductDir*>\bin directory. For more information, refer to the Connector Configurator appendix in this guide.

If the connector has not been internationalized, the only valid value for this property is `en_US`. To determine whether a specific connector has been globalized, refer to the user guide for that adapter.

The default value is `en_US`.

LogAtInterchangeEnd

The LogAtInterchangeEnd property specifies whether to log errors to the log destination of the integration broker.

Logging to the log destination also turns on e-mail notification, which generates e-mail messages for the recipient specified as the value of MESSAGE_RECIPIENT in the InterchangeSystem.cfg file when errors or fatal errors occur. For example, when a connector loses its connection to the application, if the value of LogAtInterChangeEnd is `true`, an e-mail message is sent to the specified message recipient.

This property is valid only if the value of the RespositoryDirectory property is set to <REMOTE> (the value of BrokerType is ICS).

The default value is `false`.

MaxEventCapacity

The MaxEventCapacity property specifies maximum number of events in the controller buffer. This property is used by the flow control feature.

This property is valid only if the value of the RespositoryDirectory property is set to <REMOTE> (the value of BrokerType is ICS).

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

The default value is 2147483647.

MessageFileName

The MessageFileName property specifies the name of the connector message file. The standard location for the message file is \connectors\messages in the product directory. Specify the message file name 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 connector has its own message file, see the individual adapter user guide.

The default value is `InterchangeSystem.txt`.

MonitorQueue

The MonitorQueue property specifies the logical queue that the connector uses to monitor duplicate events.

It is valid only if the value of the DeliveryTransport property is `JMS` and the value of the DuplicateEventElimination is `true`.

The default value is *<CONNECTORNAME>*`/MONITORQUEUE`

OADAutoRestartAgent

the OADAutoRestartAgent property specifies whether the connector uses the automatic and remote restart feature. This feature uses the WebSphere 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 WebSphere MQ-triggered OAD feature. see the *Installation Guide for Windows* or *for UNIX*.

This property is valid only if the value of the RespositoryDirectory property is set to <REMOTE> (the value of BrokerType is ICS).

The default value is `false`.

OADMaxNumRetry

The OADMaxNumRetry property specifies the maximum number of times that the WebSphere MQ-triggered Object Activation Daemon (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.

This property is valid only if the value of the RespositoryDirectory property is set to <REMOTE> (the value of BrokerType is ICS).

The default value is 1000.

OADRetryTimeInterval

The OADRetryTimeInterval property specifies the number of minutes in the retry-time interval for the WebSphere MQ-triggered Object Activation Daemon (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.

This property is valid only if the value of the RespositoryDirectory property is set to <REMOTE> (the value of BrokerType is ICS).

The default value is 10.

PollEndTime

The PollEndTime property specifies the time to stop polling the event queue. The format is *HH:MM*, where *HH* is 0 through 23 hours, and *MM* represents 0 through 59 minutes.

You must provide a valid value for this property. The default value is HH:MM without a value, and it must be changed.

If the adapter runtime detects:

- PollStartTime set and PollEndTime not set, or
- PollEndTime set and PollStartTime not set

it will poll using the value configured for the PollFrequency property.

PollFrequency

The PollFrequency property 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 the PollQuantity property.
- Process these objects. For some connectors, this may be partly done on separate threads, which execute asynchronously to the next polling action.
- Delay for the interval specified by the PollFrequency property.
- Repeat the cycle.

The following values are valid for this property:

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

The PollQuantity property designates the number of items from the application that the connector polls for. If the adapter has a connector-specific property for setting the poll quantity, the value set in the connector-specific property overrides the standard property value.

This property is valid only if the value of the `DeliveryTransport` property is `JMS`, and the `ContainerManagedEvents` property has a value.

An e-mail message is also considered an event. The connector actions are as follows when it is polled for e-mail.

- When it is polled once, the connector detects the body of the message, which it reads as an attachment. Since no data handler was specified for this mime type, it will then ignore the message.
- The connector processes the first BO attachment. The data handler is available for this MIME type, so it sends the business object to Visual Test Connector.
- When it is polled for the second time, the connector processes the second BO attachment. The data handler is available for this MIME type, so it sends the business object to Visual Test Connector.
- Once it is accepted, the third BO attachment should be transmitted.

PollStartTime

The PollStartTime property specifies the time to start polling the event queue. The format is *HH:MM*, where *HH* is 0 through 23 hours, and *MM* represents 0 through 59 minutes.

You must provide a valid value for this property. The default value is `HH:MM` without a value, and it must be changed.

If the adapter runtime detects:

- PollStartTime set and PollEndTime not set, or
- PollEndTime set and PollStartTime not set

it will poll using the value configured for the PollFrequency property.

RepositoryDirectory

The RepositoryDirectory property is the location of the repository from which the connector reads the XML schema documents that store the metadata for business object definitions.

If the integration broker is ICS, this value must be set to 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 is set to <*ProductDir*>\repository by default. However, it may be set to any valid directory name.

RequestQueue

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

This property is valid only if the value of the DeliveryTransport property is `JMS`.

The default value is <*CONNECTORNAME*>/REQUESTQUEUE.

ResponseQueue

The ResponseQueue property specifies 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.

This property is valid only if the value of the DeliveryTransport property is JMS.

The default value is <*CONNECTORNAME*>/RESPONSEQUEUE.

RestartRetryCount

The RestartRetryCount property specifies the number of times the connector attempts to restart itself. When this property is used for a connector that is connected in parallel, it specifies the number of times the master connector application-specific component attempts to restart the client connector application-specific component.

The default value is 7.

RestartRetryInterval

The RestartRetryInterval property specifies the interval in minutes at which the connector attempts to restart itself. When this property is used for a connector that is linked in parallel, it specifies the interval at which the master connector application-specific component attempts to restart the client connector application-specific component.

Possible values for the property range from 1 through 2147483647.

The default value is 1.

ResultsSetEnabled

The ResultsSetEnabled property enables or disables results set support when Information Integrator is active. This property can be used only if the adapter supports DB2 Information Integrator.

This property is valid only if the value of the DeliveryTransport property is JMS, and the value of BrokerType is WMQI.

The default value is false.

ResultsSetSize

The ResultsSetSize property defines the maximum number of business objects that can be returned to Information Integrator. This property can be used only if the adapter supports DB2 Information Integrator.

This property is valid only if the value of the ResultsSetEnabled property is true.

The default value is 0. This means that the size of the results set is unlimited.

RHF2MessageDomain

The RHF2MessageDomain property allows you to configure the value of the field domain name in the JMS header. When data is sent to a WebSphere message broker over JMS transport, the adapter framework writes JMS header information, with a domain name and a fixed value of `mrm`. A configurable domain name lets you track how the WebSphere message broker processes the message data.

This is an example header:

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

This property is valid only if the value of BrokerType is WMQI or WAS. Also, it is valid only if the value of the DeliveryTransport property is `JMS`, and the value of the WireFormat property is `CwXML`.

Possible values are `mrm` and `xml`. The default value is `mrm`.

SourceQueue

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

This property is valid only if the value of DeliveryTransport is `JMS`, and a value for ContainerManagedEvents is specified.

The default value is *<CONNECTORNAME>*`/SOURCEQUEUE`.

SynchronousRequestQueue

The SynchronousRequestQueue property 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 synchronous request queue and waits for a response from the broker on the synchronous response queue. The response message sent to the connector has a correlation ID that matches the ID of the original message.

This property is valid only if the value of DeliveryTransport is `JMS`.

The default value is *<CONNECTORNAME>*`/SYNCHRONOUSREQUESTQUEUE`

SynchronousRequestTimeout

The SynchronousRequestTimeout property specifies the time in milliseconds that the connector waits for a response to a synchronous request. If the response is not received within the specified time, the connector moves the original synchronous request message (and error message) to the fault queue.

This property is valid only if the value of DeliveryTransport is `JMS`.

The default value is `0`.

SynchronousResponseQueue

The SynchronousResponseQueue property delivers response messages in reply to a synchronous request from the broker to the connector framework. This queue is necessary only if the connector uses synchronous execution.

This property is valid only if the value of DeliveryTransport is `JMS`.

The default is `<CONNECTORNAME>/SYNCHRONOUSRESPONSEQUEUE`

TivoliMonitorTransactionPerformance

The TivoliMonitorTransactionPerformance property specifies whether IBM Tivoli Monitoring for Transaction Performance (ITMTP) is invoked at run time.

The default value is `false`.

WireFormat

The WireFormat property specifies the message format on the transport:

- If the value of the RepositoryDirectory property is a local directory, the value is `CwXML`.
- If the value of the RepositoryDirectory property is a remote directory, the value is `CwBO`.

WsifSynchronousRequestTimeout

The WsifSynchronousRequestTimeout property specifies the time in milliseconds that the connector waits for a response to a synchronous request. If the response is not received within the specified time, the connector moves the original synchronous request message (and an error message) to the fault queue.

This property is valid only if the value of BrokerType is `WAS`.

The default value is 0.

XMLNameSpaceFormat

The XMLNameSpaceFormat property specifies short or long namespaces in the XML format of business object definitions.

This property is valid only if the value of BrokerType is set to WMQI or WAS.

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

The topics covered in this appendix are:

Overview of Connector Configurator

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

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

If your adapter supports DB2 Information Integrator, use the WMQI options and the DB2 II standard properties (see the Notes column in the Standard Properties appendix.)

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 72).

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 73 to set up a new one.

Running connectors on UNIX

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.

Some properties in the Connector Configurator use directory paths, which default to the Windows convention for directory paths. If you use the configuration file in a UNIX environment, revise the directory paths to match the UNIX convention for these paths. Select the target operating system in the toolbar drop-list so that the correct operating system rules are used for extended validation.

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 without running System Manager and work with connector configuration files, irrespective of your broker.

To do so:

- From **Start>Programs**, click **IBM WebSphere Business Integration Adapters>IBM WebSphere Business Integration Toolset>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 78.)

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 73.
- 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
  Property Subtype
  Updated Method
  Description
- **Flags**
  Standard flags
- **Custom Flag**
  Flag

The **Property Subtype** can be selected when **Property Type** is a String. It is an optional value which provides syntax checking when you save the configuration file. The default is a blank space, and means that the property has not been subtyped.

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. Right-click on the square to the left of the Value column heading.
2. From the pop-up menu, select **Add** to display the Property Value dialog box. Depending on the property type, the dialog box allows you to enter either a value, or both a value and a range.
3. Enter 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.

Setting pathnames

Some general rules for setting pathnames are:

- The maximum length of a filename in Windows and UNIX is 255 characters.
- In Windows, the absolute pathname must follow the format [Drive:][Directory]\filename: for example, C:\WebSphereAdapters\bin\Data\Std\StdConnProps.xml
  In UNIX the first character should be /.

- Queue names may not have leading or embedded spaces.

Creating a new configuration file

When you create a new configuration file, you must name it and select an integration broker.

You also select an operating system for extended validation on the file. The toolbar has a droplist called **Target System** that allows you to select the target operating system for extended validation of the properties. The available options are: Windows, UNIX, Other (if not Windows or UNIX), and None-no extended validation (switches off extended validation). The default on startup is Windows.

To start Connector Configurator:

- In the System Manager window, select **Connector Configurator** from the **Tools** menu. Connector Configurator opens.
- In stand-alone mode, launch Connector Configurator.

To set the operating system for extended validation of the configuration file:

- Pull down the **Target System:** droplist on the menu bar.
- Select the operating system you are running on.

Then 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.
- Complete 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. Set the operating system for extended validation of the configuration file using the **Target System:** droplist on the menu bar (see "Creating a new configuration file" above).
2. Click **File>New>Connector Configuration**.
3. 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**.

4. 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 complete the definition now, or you can save the file and complete the fields later.
5. 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.
6. 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 connector properties associated with the selected broker. The table shows **Property name**, **Value**, **Type**, **Subtype** (if the Type is a string), **Description**, and **Update Method**.
3. You can save the file now or complete the remaining configuration fields, as described in "Specifying supported business object definitions" on page 81..
4. 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 you created the configuration file, you used the **Target System** droplist that allows you to select the target operating system for extended validation of the properties.

   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.

   If you have elected to use the extended validation feature by selecting a value of Windows, UNIX or Other from the **Target System** droplist, the system will validate the property subtype s well as the type, and it displays a warning message if the validation fails.

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)
- Security

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.

   Note: If the property has a Type of String, it may have a subtype value in the Subtype column. This subtype is used for extended validation of the property.
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.

To get more information on a particular standard property, left-click the entry in the Description column for that property in the Standard Properties tabbed sheet. If you have Extended Help installed, an arrow button will appear on the right. When you click on the button, a Help window will open and display details of the standard property.

Note: If the hot button does not appear, no Extended Help was found for that property.

If installed, the Extended Help files are located in <*ProductDir*>\bin\Data\Std\Help\<*RegionalSetting*>\.

Setting connector-specific configuration properties

For connector-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.

   Note: If the property has a Type of String, you can select a subtype from the Subtype droplist. This subtype is used for extended validation of the property.
3. To encrypt a property, select the **Encrypt** box.

4. To get more information on a particular property, left-click the entry in the Description column for that property. If you have Extended Help installed, a hot button will appear. When you click on the hot button, a Help window will open and display details of the standard property.

   Note: If the hot button does not appear, no Extended Help was found for that property.
5. Choose to save or discard changes, as described for "Setting standard connector properties" on page 80.

If the Extended Help files are installed and the AdapterHelpName property is blank, Connector Configurator will point to the adapter-specific Extended Help files located in <*ProductDir*>\bin\Data\App\Help\<*RegionalSetting*>\. Otherwise, Connector Configurator will point to the adapter-specific Extended Help files located in <*ProductDir*>\bin\Data\App\Help\<*AdapterHelpName*>\<*RegionalSetting*>\. See the AdapterHelpName property described in the Standard Properties appendix.

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 Properties appendix, under "Configuration property values overview" on page 48.

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 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)

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 Binding**

  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** tab enables you to configure messaging properties. 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.

Validating messaging queues

Before you can validate a messaging queue, you must:

- Make sure that WebSphere MQ Series is installed.
- Create a messaging queue with channel and port on the host machine.
- Set up a connection to the host machine.

To validate the queue, use the Validate button to the right of the Messaging Type and Host Name fields on the Messaging tab.

Security (ICS)

You can use the **Security** tab in Connector Configurator to set various privacy levels for a message. You can only use this feature when the DeliveryTransport property is set to JMS.

By default, Privacy is turned off. Check the **Privacy** box to enable it.

The **Keystore Target System Absolute Pathname** is:

- For Windows:
  `<ProductDir>\connectors\security\<connectorname>.jks`
- For UNIX:
  `opt/IBM/WebSphereAdapters/connectors/security/<connectorname>.jks`

This path and file should be on the system where you plan to start the connector, that is, the target system.

You can use the Browse button at the right only if the target system is the one currently running. It is greyed out unless **Privacy** is enabled and the **Target System** in the menu bar is set to Windows.

The **Message Privacy Level** may be set as follows for the three messages categories (All Messages, All Administrative Messages, and All Business Object Messages):

- "" is the default; used when no privacy levels for a message category have been set.
- none
  Not the same as the default: use this to deliberately set a privacy level of none for a message category.
- integrity
- privacy
- integrity_plus_privacy

The **Key Maintenance** feature lets you generate, import and export public keys for the server and adapter.

- When you select **Generate Keys**, the Generate Keys dialog box appears with the defaults for the keytool that will generate the keys.
- The keystore value defaults to the value you entered in **Keystore Target System Absolute Pathname** on the Security tab.
- When you select OK, the entries are validated, the key certificate is generated and the output is sent to the Connector Configurator log window.

Before you can import a certificate into the adapter keystore, you must export it from the server keystore. When you select **Export Adapter Public Key**, the Export Adapter Public Key dialog box appears.

- The export certificate defaults to the same value as the keystore, except that the file extension is <filename>.cer.

When you select **Import Server Public Key**, the Import Server Public Key dialog box appears.

- The import certificate defaults to <*ProductDir*>\bin\ics.cer (if the file exists on the system).
- The import Certificate Association should be the server name. If a server is registered, you can select it from the droplist.

The **Adapter Access Control** feature is enabled only when the value of DeliveryTransport is IDL. By default, the adapter logs in with the guest identity. If the **Use guest identity** box is not checked, the **Adapter Identity** and **Adapter Password** fields are enabled.

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 in the properties window 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. Common Event Infrastructure

WebSphere Business Integration Server Foundation includes the Common Event Infrastructure Server Application, which is required for Common Event Infrastructure to operate. The WebSphere Application Server Foundation can be installed on any system (it does not have to be the same machine on which the adapter is installed.)

The WebSphere Application Server Application Client includes the libraries required for interaction between the adapter and the Common Event Infrastructure Server Application. You must install WebSphere Application Server Application Client on the same system on which you install the adapter. The adapter connects to the WebSphere Application Server (within the WebSphere Business Integration Server Foundation) by means of a configurable URL.

Common Event Infrastructure support is available using any integration broker supported with this release.

Required software

In addition to the software prerequisites required for the adapter, you must have the following installed for Common Event Infrastructure to operate:

- WebSphere Business Integration Server Foundation 5.1.1
- WebSphere Application Server Application Client 5.0.2, 5.1, or 5.1.1.

  (WebSphere Application Server Application Client 5.1.1 is provided with WebSphere Business Integration Server Foundation 5.1.1.)

Note: Common Event Infrastructure is not supported on any HP-UX or Linux platform.

Enabling Common Event Infrastructure

Common Event Infrastructure functionality is enabled with the standard properties `CommonEventInfrastructure` and `CommonEventInfrastructureContextURL`, configured with Connector Configurator. By default, Common Event Infrastructure is not enabled. The `CommonEventInfrastructureContextURL` property enables you to configure the URL of the Common Event Infrastructure server.(Refer to the "Standard Properties" appendix of this document for more information.)

Obtaining Common Event Infrastructure adapter events

If Common Event Infrastructure is enabled, the adapter generates Common Event Infrastructure events that map to the following adapter events:

- Starting the adapter
- Stopping the adapter
- An application response to a timeout from the adapter agent
- Any `doVerbFor` call issued from the adapter agent
- A `gotApplEvent` call from the adapter agent

For another application (the "consumer application") to receive the Common Event Infrastructure events generated by the adapter, the application must use the

Common Event Infrastructure event catalog to determine the definitions of appropriate events and their properties. The events must be defined in the event catalog for the consumer application to be able to consume the sending application's events.

The "Common Event Infrastructure event catalog definitions" appendix of this document contains XML format metadata showing, for WebSphere Business Information adapters, the event descriptors and properties the consumer application should search for.

For more information

For more information about Common Event Infrastructure, refer to the Common Event Infrastructure information in the WebSphere Business Integration Server Foundation documentation, available at the following URL:

http://publib.boulder.ibm.com/infocenter/ws51help

For sample XML metadata showing the adapter-generated event descriptors and properties a consumer application should search for, refer to"Common Event Infrastructure event catalog definitions."

Common Event Infrastructure event catalog definitions

The Common Event Infrastructure event catalog contains event definitions that can be queried by other applications. The following are event definition samples, using XML metadata, for typical adapter events. If you are writing another application, your application can use event catalog interfaces to query against the event definition. For more information about event definitions and how to query them, refer to the Common Event Infrastructure documentation that is available from the online IBM WebSphere Server Foundation Information Center.

For WebSphere Business Integration adapters, the extended data elements that need to be defined in the event catalog are the keys of the business object. Each business object key requires an event definition. So for any given adapter, various events such as start adapter, stop adapter, timeout adapter, and any `doVerbFor` event (create, update, or delete, for example) must have a corresponding event definition in the event catalog.

The following sections contain examples of the XML metadata for start adapter, stop adapter, and event request or delivery.

XML format for "start adapter" metadata

```
<eventDefinition name="startADAPTER"
      parent="event">
    <property name ="creationTime" //Comment: example value would be
 "2004-05-13T17:00:16.319Z"
         required="true" />
    <property name="globalInstanceId" //Comment: Automatically generated
 by Common Event Infrastructure
         required="true"/>
    <property name="sequenceNumber"    //Comment: Source defined number
for messages to be sent/sorted logically
         required="false"/>
    <property name="version"    //Comment: Version of the event
         required="false"
         defaultValue="1.0.1"/>
```

```
    <property name="sourceComponentId"
         path="sourceComponentId"
         required="true"/>
    <property name="application"   //Comment: The name#version of the
source application generating the event. Example is "SampleConnector#3.0.0"
         path="sourceComponentId/application"         required="false"/>
    <property name="component"   //Comment: This will be the name#version
 of the source component.
         path="sourceComponentId/component"
         required="true"
         defaultValue="ConnectorFrameWorkVersion#4.2.2"/>
    <property name="componentIdType"    //Comment: specifies the format
and meaning of the component
          path="sourceComponentId/componentIdType"
          required="true"
          defaultValue="Application"/>
    <property name="executionEnvironment"
 //Comment: Identifies the environment the application is running
 in...example is "Windows 2000#5.0"
          path="sourceComponentId/executionEnvironment"
          required="false" />
     <property name="location"    //Comment: The value of this is the
 server name...example is "WQMI"
          path="sourceComponentId/location"
          required="true"/>
    <property name="locationType"   //Comment specifies the format and
     meaning of the location
          path="sourceComponentId/locationType"
          required="true"
          defaultValue="Hostname"/>
     <property name="subComponent"     //Comment:further distinction
of the logical component
          path="sourceComponentId/subComponent"
          required="true"
          defaultValue="AppSide_Connector.AgentBusinessObjectManager"/>
     <property name="componentType"      //Comment: well-defined name
used to characterize all instances of this component
          path="sourceComponentId/componentType"
          required="true"
          defaultValue="ADAPTER"/>
     <property name="situation"   //Comment: Defines the type of
 situation that caused the event to be reported
          path="situation"
          required="true"/>
     <property name="categoryName="    //Comment: Specifies the type
of situation for the event
         path="situation/categoryName"
         required="true"
         defaultValue="StartSituation"/>
    <property name="situationType"    //Comment: Specifies the type
of situation and disposition of the event
         path="situation/situationType"
         required="true"
    <property name="reasoningScope" //Comment: Specifies the scope
 of the impact of the event
         path="situation/situationType/reasoningScope"
         required="true"
         permittedValue="INTERNAL"
         permittedValue="EXTERNAL"/>
    <property name="successDisposition" //Comment: Specifies the
 success of event
         path="situation/situationType/successDisposition"
         required="true"
         permittedValue="SUCCESSFUL"
         permittedValue="UNSUCCESSFUL" />
    <property name="situationQualifier"  //Comment: Specifies the
 situation qualifiers for this event
```

```
        path="situation/situationType/situationQualifier"
        required="true"
        permittedValue="START_INITIATED"
        permittedValue="RESTART_INITIATED"
        permittedValue="START_COMPLETED" />
</eventDefinition>
```

XML format for ″stop adapter″ metadata

The metadata for “stop adapter” is the same as that for “start adapter” with the following exceptions:

- The default value for the `categoryName` property is `StopSituation`:

```
<property name="categoryName="
 //Comment: Specifies the type
 of situation for the event
             path="situation/categoryName"
             required="true"
             defaultValue="StopSituation"/>
```

- The permitted values for the `situationQualifier` property differ and are as follows for “stop adapter”:

```
<property name="situationQualifier"
 //Comment: Specifies the situation qualifiers for this event
          path="situation/situationType/situationQualifier"
          required="true"
          permittedValue="STOP_INITIATED"
          permittedValue="ABORT_INITIATED"
          permittedValue="PAUSE_INITIATED"
          permittedValue="STOP_COMPLETED"
 />
```

XML format for “timeout adapter” metadata

The metadata for “timeout adapter” is the same as that for “start adapter” and “stop adapter” with the following exceptions:

- The default value for the `categoryName` property is `ConnectSituation`:

```
<property name="categoryName="
 //Comment: Specifies the type
 of situation for the event
             path="situation/categoryName"
             required="true"
             defaultValue="ConnectSituation"/>
```

- The permitted values for the `situationQualifier` property differ and are as follows for “timeout adapter”:

```
<property name="situationQualifier"  //Comment: Specifies
 the situation qualifiers for this event
          path="situation/situationType/situationQualifier"
          required="true"
          permittedValue="IN_USE"
          permittedValue="FREED"
          permittedValue="CLOSED"
          permittedValue="AVAILABLE"
 />
```

XML format for ″request″ or ″delivery″ metadata

At the end of this XML format are the extended data elements. The extended data elements for adapter request and delivery events represent data from the business object being processed. This data includes the name of the business object, the key (foreign or local) for the business object, and business objects that are children of parent business objects. The children business objects are then broken down into the same data as the parent (name, key, and any children business objects). This data is represented in an extended data element of the event definition. This data will change depending on which business object, which keys, and which child business objects are being processed. The extended data in this event definition is just an example and represents a business object named `Employee` with a key `EmployeeId` and a child business object `EmployeeAddress` with a key `EmployeeId`. This pattern could continue for as much data as exists for the particular business object.

```
<eventDefinition name="createEmployee"     //Comment: This
 extension name is always the business object verb followed by the business
 object name
          parent="event">
    <property name ="creationTime"  //Comment: example value would be
"2004-05-13T17:00:16.319Z"
         required="true" />
    <property name="globalInstanceId" //Comment: Automatically generated
 by Common Event Infrastructure
         required="true"/>
    <property name="localInstanceId"    //Comment: Value is business
 object verb+business object name+#+app name+ business object identifier
         required="false"/>
    <property name="sequenceNumber"    //Comment: Source defined number
for messages to be sent/sorted logically
         required="false"/>
    <property name="version"  //Comment: Version of the event...value is
 set to 1.0.1
         required="false"
         defaultValue="1.0.1"/>
    <property name="sourceComponentId"
         path="sourceComponentId"
         required="true"/>
    <property name="application"    //Comment: The name#version of the
 source application generating the event...example is
 "SampleConnector#3.0.0"
         path="sourceComponentId/application"
         required="false"/>
    <property name="component"   //Comment: This will be the name#version
of the source component.
         path="sourceComponentId/component"
         required="true"
         defaultValue="ConnectorFrameWorkVersion#4.2.2"/>
    <property name="componentIdType"     //Comment: specifies the format
 and meaning of the component
         path="sourceComponentId/componentIdType"
         required="true"
         defaultValue="Application"/>
    <property name="executionEnvironment" //Comment: Identifies the
 environment#version the app is running in...example is "Windows 2000#5.0"
         path="sourceComponentId/executionEnvironment"
         required="false" />
    <property name="instanceId" //Comment: Value is business object
  verb+business object name+#+app name+ business object identifier
          path="sourceComponentId/instanceId"
          required="false"
    <property name="location"   //Comment: The value of this is the
server name...example is "WQMI"
          path="sourceComponentId/location"
```

```
          required="true"/>
     <property name="locationType" //Comment specifies the format and
 meaning of the location
          path="sourceComponentId/locationType"
          required="true"
          defaultValue="Hostname"/>
     <property name="subComponent"  //Comment:further distinction of the
 logical component-in this case the value is the name of the business
 object
          path="sourceComponentId/subComponent"
          required="true"/>
     <property name="componentType"      //Comment: well-defined name used
 to characterize all instances of this component
          path="sourceComponentId/componentType"
          required="true"
          defaultValue="ADAPTER"/>
     <property name="situation" //Comment: Defines the type of
situation that caused the event to be reported
          path="situation"
          required="true"/>
    <property name="categoryName"    //Comment: Specifies the type
 of situation for the event
         path="situation/categoryName"
         required="true"
         permittedValue="CreateSituation"
         permittedValue="DestroySituation"
         permittedValue="OtherSituation" />
    <property name="situationType"    //Comment: Specifies the type
of situation and disposition of the event
         path="situation/situationType"
         required="true"
    <property name="reasoningScope" //Comment: Specifies the scope
of the impact of the event
         path="situation/situationType/reasoningScope"
         required="true"
         permittedValue="INTERNAL"
         permittedValue="EXTERNAL"/>
    <property name="successDisposition" //Comment: Specifies the
 success of event
         path="situation/situationType/successDisposition"
         required="true"
         permittedValue="SUCCESSFUL"
         permittedValue="UNSUCCESSFUL" />
    <extendedDataElements name="Employee" //Comment: name of business
 object itself
             type="noValue"
             <children name="EmployeeId"
                  type="string"/>  //Comment: type is one of the
 permitted values within Common Event Infrastructure documentation
             <children name="EmployeeAddress"
                  type="noValue"/>
                    <children name="EmployeeId"
                        type="string"/>
                     -
                     -
                     -
    </extendedDataElements
</eventDefinition>
```

Appendix D. Application Response Measurement

This adapter is compatible with the Application Response Measurement application programming interface (API), an API that allows applications to be managed for availability, service level agreements, and capacity planning. An ARM-instrumented application can participate in IBM Tivoli Monitoring for Transaction Performance, allowing collection and review of data concerning transaction metrics.

Application Response Measurement instrumentation support

This adapter is compatible with the Application Response Measurement application programming interface (API), an API that allows applications to be managed for availability, service level agreements, and capacity planning. An ARM-instrumented application can participate in IBM Tivoli Monitoring for Transaction Performance, allowing collection and review of data concerning transaction metrics.

Required software

In addition to the software prerequisites required for the adapter, you must have the following installed for ARM to operate:

- WebSphere Application Server 5.0.1 (contains the IBM Tivoli Monitoring for Transaction Performance server). This does not have to be installed on the same system as the adapter.
- IBM Tivoli Monitoring for Transaction Performance v. 5.2 Fixpack 1. This must be installed on the same system on which the adapter is installed and configured to point to the system on which the IBM Tivoli Monitoring for Transaction Performance server resides.

Application Response Measurement support is available using any integration broker supported with this release.

Note: Application Response Measurement instrumentation is supported on all operating systems supported with this IBM WebSphere Business Integration Adapters release *except* HP-UX (any version) and Red Hat Linux 3.0.

Enabling Application Response Measurement

ARM instrumentation is enabled via by setting the standard property `TivoliMonitorTransactionPerformance` in Connector Configurator to "True." By default ARM support is not enabled. (Refer to the ″Standard Properties″ appendix of this document for more information.)

Transaction monitoring

When ARM is enabled, the transactions that are monitored are service events and event deliveries. The transaction is measured from the start of a service request or event delivery to the end of the service request or event delivery. The name of the transaction displayed on the Tivoli Monitoring for Transaction Performance console will start with either `SERVICE REQUEST` or `EVENT DELIVERY`. The next part of the name will be the business object verb (such as `CREATE`, `RETRIEVE`, `UPDATE` or `DELETE`). The final part of the name will be the business object name such as "`EMPLOYEE`."

For example, the name of a transaction for an event delivery for creation of an employee might be `EVENT DELIVERY CREATE EMPLOYEE`. Another might be `SERVICE REQUEST UPDATE ORDER`.

The following metrics are collected by default for each type of service request or event delivery:

- Minimum transaction time
- Maximum transaction time
- Average transaction time
- Total transaction runs

You (or the system administrator of the WebSphere Application Server) can select which of these metrics to display, for which adapter events, by configuring Discovery Policies and Listener Policies for particular transactions from within the Tivoli Monitoring for Transaction Performance console. (Refer to "For more information.")

For more information

Refer to the IBM Tivoli Monitoring for Transaction Performance documentation for more information. In particular, refer to the *IBM Tivoli Monitoring for Transaction Performance User's Guide* for information about monitoring and managing the metrics generated by the adapter.

Appendix E. Notices

This information was developed for products and services offered in the U.S.A.

IBM may not offer the products, services, or features discussed in this document in other 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 grant 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.

For license inquiries regarding double-byte (DBCS) information, contact the IBM Intellectual Property Department in your country or send inquiries, in writing, to:

IBM World Trade Asia Corporation Licensing
2-31 Roppongi 3-chome, Minato-ku
Tokyo 106-0032, Japan

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 the 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 Corporation
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 measurements 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 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.

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

This information contains examples of data and reports used in daily business operations. To illustrate them as completely as possible, the examples 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.

COPYRIGHT LICENSE:

This information contains sample application programs in source language, which illustrate 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.

If you are viewing this information softcopy, the photographs and color illustrations may not appear.

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:

i5/OS
IBM
the IBM logo
AIX
AIX 5L
CICS
CrossWorlds
DB2
DB2 Universal Database
Domino
HelpNow
IMS
Informix
iSeries
Lotus
Lotus Notes
MQIntegrator
MQSeries
MVS
Notes
OS/400
Passport Advantage
pSeries
Redbooks
SupportPac
WebSphere
z/OS

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

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

Intel, Intel logo, Intel Inside, Intel Inside logo, Intel Centrino, Intel Centrino logo, Celeron, Intel Xeon, Intel SpeedStep, Itanium, and Pentium are trademarks or registered trademarks of Intel Corporation or its subsidiaries in the United States and other countries.

UNIX is a registered trademark of The Open Group in the United States and other countries.

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

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

This product includes software developed by the Eclipse Project (http://www.eclipse.org/).

WebSphere Business Integration Adapter Framework, version 2.6.0.3

WebSphere Business Integration Adapter Framework V2.6

Printed in USA