IBM WebSphere Business Integration Adapters

Adapter for JD Edwards OneWorld User Guide

Version 2.0.x

IBM WebSphere Business Integration Adapters

Adapter for JD Edwards OneWorld User Guide

Version 2.0.x

Note!
Before using this information and the product it supports, read the information in "Notices" on page 101.

22December2006

This edition of this document applies to IBM WebSphere Business Integration Adapter for JD Edwards OneWorld, version 2.0.4.

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

IBM® WebSphere®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 JD Edwards OneWorld.

Audience

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

Prerequisites for this document

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

Related documents

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

You can download, install, and view the documentation at the following site:

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

The documentation set consists of Portable Document Format (PDF) files and files in HTML format. To read it, you need an HTML browser such as Netscape Navigator (version 4.7 or higher) or Internet Explorer (version 5.5 or higher) and Adobe Acrobat Reader (Version 4.0.5 or higher). For the latest version of Adobe Acrobat Reader for your platform, go to the Adobe Web site (http://www.adobe.com).

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:

Typographic convention	Description
`courier font`	Indicates a literal value, such as a command name, filename, information that you type, or information that the system prints on the screen.
bold	Indicates a new term the first time that it appears.
italic, *`italic`*	Indicates a variable name or a cross-reference.
blue outline	A blue outline, which is visible only when you view the manual online, indicates a cross-reference hyperlink. Click inside the outline to jump to the object of the reference.
{ }	In a syntax line, curly braces surround a set of options from which you must choose one and only one.
`[ ]`	In a syntax line, square brackets surround an optional parameter.
`...`	In a syntax line, ellipses indicate a repetition of the previous parameter. For example, `option[,...]` means that you can enter multiple, comma-separated options.
`< >`	In a naming convention, angle brackets surround individual elements of a name to distinguish them from each other, as in `<server_name><`*`connector_name`*`>tmp.log`.
/, \	In this document, backslashes (\) are used as the convention for directory paths. For UNIX installations, substitute slashes (/) for backslashes. All product pathnames are relative to the directory where the product is installed on your system.
`%text%` and *`$text`*	Text within percent (%) signs indicates the value of the Windows `text` system variable or user variable. The equivalent notation in a UNIX environment is *`$text`*, indicating the value of the text UNIX environment variable.
ProductDir	Represents the directory where the product is installed.

New in this release

New in release 2.0.4

This release incorporates fix pack and product changes occuring since the last release of this document. Unless otherwise noted, the features described are new as of release 2.0.4.

With fix pack 2.0.1 of the Adapter for JD Edwards OneWorld, the following issues are resolved:

- The adapter supports JD Edwards OneWorld Application verion 8.93 (PeopleSoft Enterprise One).
- The object discovery agent (ODA) works with Enterprise One version 8.9 and OneWorld version 9.0. The ODA successfully generates business object definitions, which are displayed when you select the the business function jar file. The spaces in table names are replaced with underscores (_).

With version 2.0.4, the adapter supports PeopleSoft EnterpriseOne Tools version 8.95.

This document provides URLs that show how to create, package, and deploy custom business functions for event notification. For more information, see "Creating, packaging, and deploying custom business functions" on page 41.

For XML business object functions, attributes clause_type and clause_seq are defined. For more information, see "Attribute-level ASI" on page 42.

You can download and install a JD Edwards OneWorld version 8.93 event store. For more information, see "Installing and configuring IBM event store" on page 19.

For event notification, the adapter supports business functions only, not XML List operations. For request processing, you can implement both business function and XML List objects to work with the adapter. For more information and an overview, see "Connector overview" on page 2.

This document now lists and explains event status codes. For more more information, see "Event status codes" on page 52.

The connector-specific property, UseDefaults, is not used by any version of the adapter.

If you are running the adapter with WebSphere BusinessIntegration Adapter Framework 2.6, you must modify startup scripts for the connector and for the object discovery agent. Otherwise, the adapter may not work with the new framework. For more information, see "Modify startup script for Adapter Framework 2.6" on page 15.

This document now describes the event business object structure. This information helps you populate an event to the event table. For more information, see "Population of an event into the event table" on page 21.

For Windows 2000 platforms, you must modify the startup script to avoid invocation errors. For more information and suggested modifications, see “Modify startup script for Windows 2000” on page 15.

The business object table type, TABLE_CONVERSION, is not supported.

New in release 2.0.x

Version 2.0.x of the Adapter for JD Edwards OneWorld supports XML List APIs.

New in release 1.0.0

Version 1.0.0 is the first release of the Adapter for JD Edwards OneWorld.

Chapter 1. Overview

This chapter provides an overview of the connector component of the WebSphere Business Integration Adapter for JD Edwards OneWorld and contains the following sections:

Terminology

The following terms are used in this guide:

- **ASI (Application-Specific Information)** Metadata tailored to a particular application or technology. ASI exists at both the attribute, verb, and business object level of a business object. See also **Verb ASI**.
- **BF (Business Function)** A collection of C functions and their associated data structures, logically grouped to perform a specific task. Regular business functions perform simple tasks, such as tax calculation or account number validation. Master business functions perform tasks that are more complex and can call several regular business functions.
- **BO (Business Object)** A set of attributes that represent a business entity (such as Employee) and an action on the data (such as a create or update operation). Components of the WebSphere business integration system use business objects to exchange information and trigger actions.
- **BO (Business Object) handler** A connector component that contains methods that interact with an application and that transforms request business objects into application operations.
- **Connector agent** A component of the connector that processes service call requests from the Integration broker as well as event notifications from OneWorld.
- **Connection object** A connection is a reference to an application that can contain state information. For every instance of a connection on the adapter side, there is a corresponding object on the JD Edwards OneWorld side. The business object handler creates connections when required, up to the maximum size of the value of the pool size property. The new connections are maintained in the pool and are re-used by multiple business object executions.
- **Connection pool** A repository used to store and retrieve connection objects.
- **GenJava** A utility provided by JD Edwards OneWorld to generate Java wrappers for the business functions running as part of the OneWorld server. GenJava creates Java Class files for the interface classes and associated data structures, compiles the generated Java files, generates Java docs, and packages them into two jar files: one for Java classes and the other for Java doc.
- **Interoperability framework** Allows seamless sharing of function and information between disparate software applications. Includes business function wrappers that provide a single point of access to major and minor business functions. Also includes master business function wrappers.

- **Java objects** Wrappers, implemented in Java, around OneWorld business functions and data structures. Java objects provide a one-to-one correspondence with OneWorld business functions.
- **ODA (object discovery agent)** A tool that automatically generates a business object definition by examining specified entities within the application and "discovering" the elements of these entities that correspond to business object attributes. When you install the adapter, the ODA is automatically installed.
- **Verb ASI (application-specific information)** For a given verb, the verb ASI specifies how the connector should process the business object when that verb is active. It can contain the name of the method to call to process the current request business object.
- **XML List** A retrieval interface to OneWorld, with which you can fetch data from a table or from a predefined table conversion process.

Connector overview

The connector for JD Edwards OneWorld is a runtime component of the WebSphere Business Integration Adapter for JD Edwards OneWorld.

WebSphere Business Integration Adapter for JD Edwards OneWorld V2.0 enables bidirectional, real-time integration between JD Edwards OneWorld Xe 8.x as well as other E-commerce, CRM, supply chain, and ERP applications. This adapter is synchronous and entirely Java-based. The JD Edwards adapter interacts with OneWorld through OneWorld Java APIs and includes an event notification mechanism through OneWorld triggers.

The JD Edwards OneWorld Adapter includes a connector, message files, configuration tools, and an Object Discovery Agent (ODA). The connector allows the WebSphere integration broker to exchange data between business objects and their corresponding OneWorld objects running on a OneWorld server.

The primary role of the generic OneWorld adapter is to act as an agent to facilitate communication and data exchange between a OneWorld server and the integration broker. The adapter is developed in Java and uses the OneWorld component jar files generated by the GenJava interface tool, provided by OneWorld.

OneWorld objects are business functions that run as part of the OneWorld server. The WebSphere Business Integration Adapter for OneWorld uses the OneWorld Java connector to invoke business functions.

The XML List API is another way of retrieving a list of records from EnterpriseOne. For request processing, you can implement both business function and XML List objects to work with the adapter. However, for event notification, the adapter supports business functions only, not XML List operations.

Connector architecture

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

- Receives and sends business objects

- Manages the exchange of startup and administrative messages

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

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

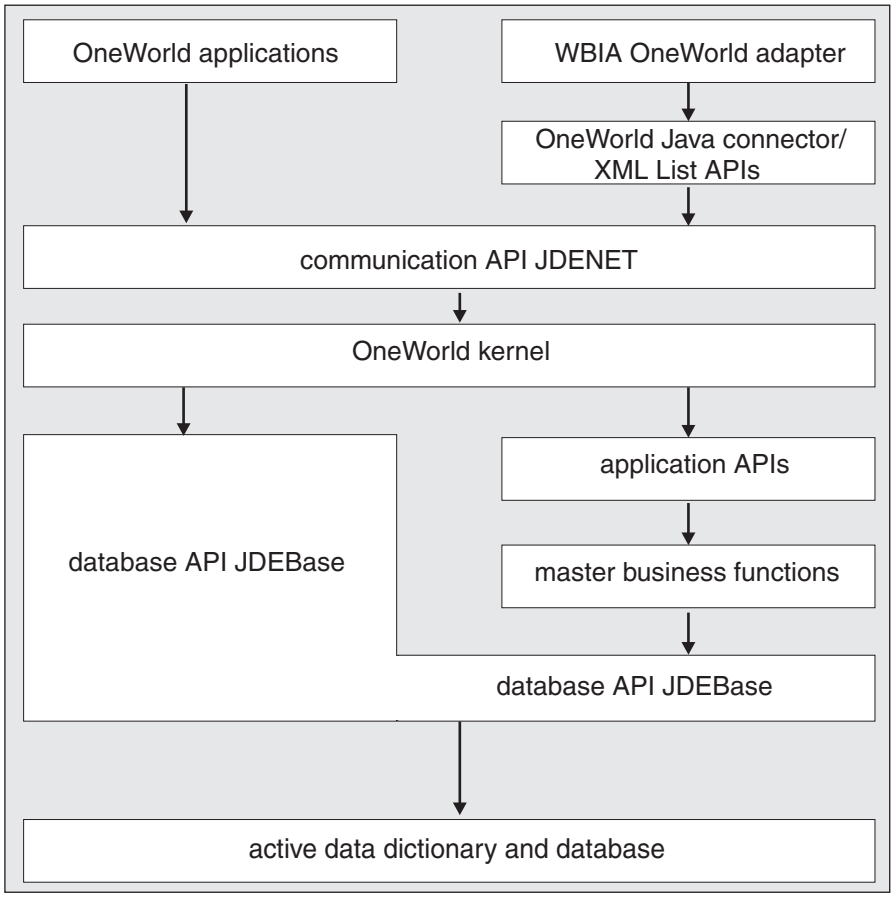

Figure 1. Connector architecture

Business functions

OneWorld business functions perform specific tasks, such as journal entry transactions, calculating deprecation, and sales order transactions. There are two types of business functions. Regular business functions perform simple tasks, such as tax calculation or account number validation. Master business functions perform tasks that are more complex, and can call several regular business functions to perform those tasks.

The interoperability framework includes business function wrappers that provide a single point of access to major and minor business functions. It also includes master business function wrappers.

When there are no business functions available to call a particular business object, you can use XML List. The following diagram illustrates the flow of an XML call.

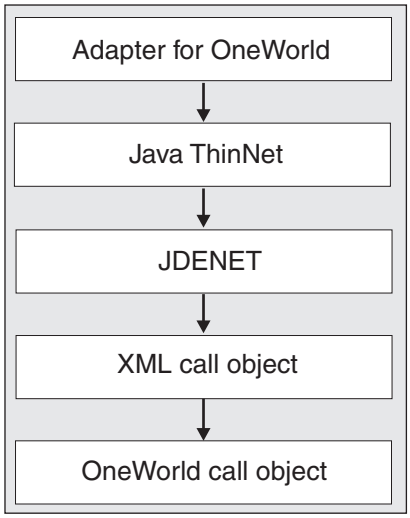

Figure 2. XML List call flow

I

When an XML call occurs, the following steps happen:

1. An interoperability client sends an XML document to OneWorld.
2. The client uses APIs defined in C++ or Java ThinNet to send the XML document to JDENET.
3. The ThinNet uses multi-threaded architecture to perform load balancing and to manage multiple XML documents simultaneously.
4. If the interoperability client sends a call object (only used for synchronous requests like handling service calls), the adapter will not use the XML transaction APIs. An XML document, received from the adapter for OneWorld, processes the request using JDENET. The following steps describe the process:
   a. The XML document creates a socket connection.
   b. Generates a JDENET message.
   c. Sends the JDENET message.
   d. Receives a JDENET message.
   e. Unpacks the response data.
   f. Closes the socket connection.
   g. Passes out the response data to an XML response file.
   h. Returns the generated response file to the adapter for OneWorld.
5. If the interoperability client does not send a call object, then it sends an XML transaction API (usually used for asynchronous requests).

The following diagram illustrates the process flow for ThinNet used in `XMLCallObject`.

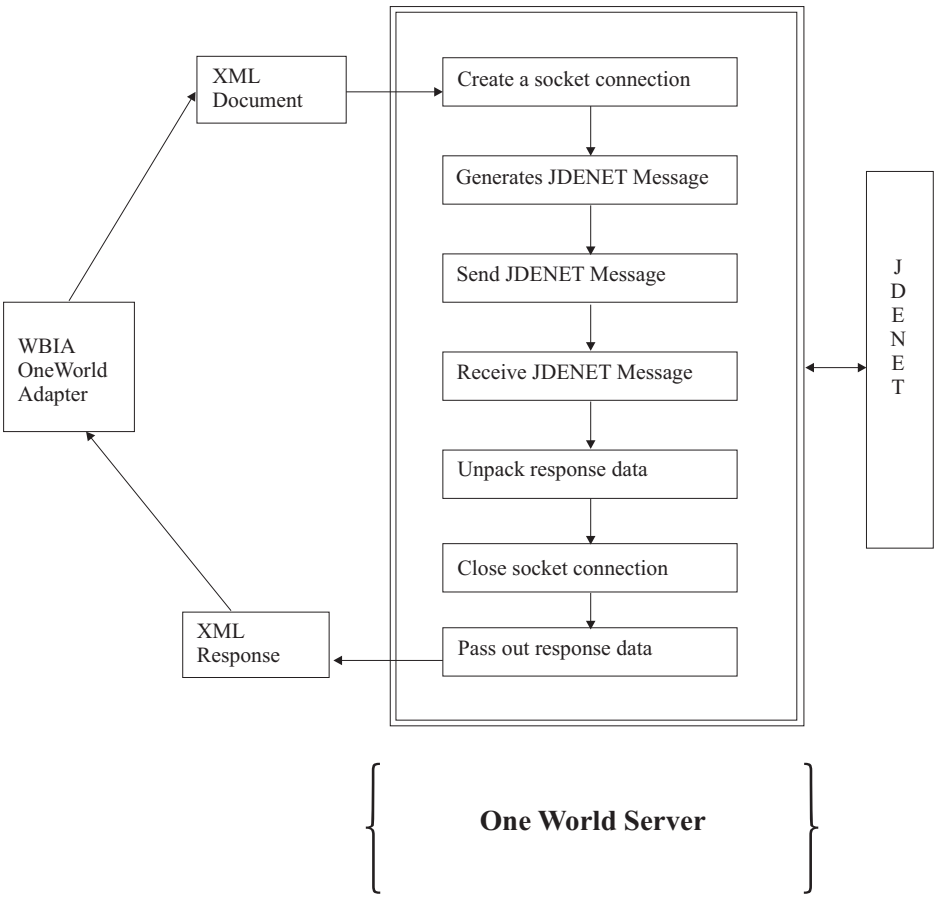

Figure 3. Process flow

The following diagram illustrates the adapter architecture.

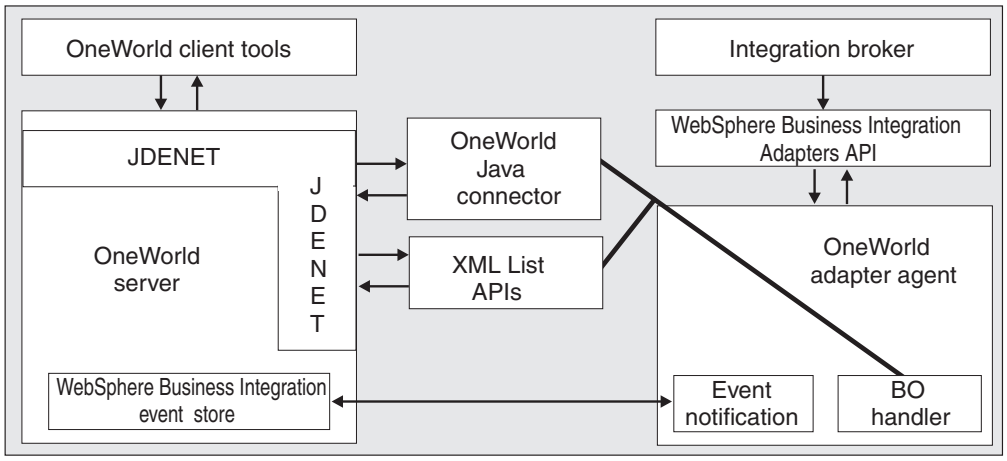

Figure 4. Architectural diagram

OneWorld supports several APIs for communication with third-party applications such as Java, COM, Oracle Database Applications, XML, and Table Conversion.

The adapter uses the Java APIs for invoking business functions in OneWorld. The business objects map to business function classes or objects.

A summary of the implementation is as follows:

1. Prepare an iJDEScript file for running the GenJava process. Refer to "Sample GenJava script file" on page 44.
2. Run the GenJava utility to generate the jar files for OneWorld objects.
3. Run the ODA tool to generate business objects for the business functions and XML List. Refer to "Running Business Object Designer" on page 24.
4. Set the key fields and the foreign key fields if you need to map data between two business objects.
5. Add the business objects to the adapter configuration file. Refer to "Startup scripts for Adapter Framework 2.6" on page 97.
6. Start the adapter. Refer to "Starting the connector" on page 19.

Request processing

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

When the connector framework passes the request business object to `doVerbFor()`, if the business object maps to an interface object, the business object handler reads the verb ASI and translates it into a series of callable functions. You can give these functions specific semantic meaning through the Object Discovery Agent (ODA) running in Business Object Designer. For details about using the ODA to assign a method call sequence to a verb, see Chapter 4, "Creating and modifying business objects," on page 23. The order in which the calls are made is critical to the successful processing of the object.

In the case of an interface business object with a blank verb ASI, the business object handler searches for a business function attribute with populated parameters and calls that business function. Only one method can be populated; otherwise, if multiple methods are populated but the verb ASI is blank, the connector logs an error and returns a `FAIL` code. For details about error processing, see "Error handling" on page 49.

If the business object maps to a business function object, the business object handler invokes the specific business function with the data specified in the business object.

If retrieval business functions are not available for specific business objects, you can use the XML List API for retrieval functionality.

The connector does not support any specific verbs for the interface business objects, but using the ODA, the user can configure verbs for a business object. The standard verbs used by WebSphere Business Integration are `Create`, `Retrieve`, `Update`, and `Delete`.

For business function business objects, the ODA generates a default verb, `Execute`. The verb ASI is not required for these business objects.

To support special access permissions for a business object, a meta business object is defined with the name ACCESS_LEVEL. The ACCESS_LEVEL business object has two attributes, Username and Password, both of type String. A OneWorld business object that has special access rules and cannot be accessed through the Username specified in the connector configuration file, would have this business object (ACCESS_LEVEL) as a child business object with single cardinality. You must add this child business object only to the top-level business object in a doVerbFor call. All the child business objects for this top-level business object must be accessible through the Username specified in the ACCESS_LEVEL child business object.

The business object handler checks to see if the top-level business object has a child business object of type ACCESS_LEVEL. If yes, and the value for the Username attribute in that business object is different from what is being used by the adapter, then it opens a new connection for processing this business object using the value for attributes Username and Password of the child business object. The connection closes after the processing of the business object is complete.

If the top-level business object does not have a child business object of type ACCESS_LEVEL, or the value of the Username attribute is the same as the Username specified in the adapter properties, the business object handler fetches a connection object from the pool.

If there is no available connection object and the maximum limit of the pool size is not reached, the business object handler creates a new connection object in the pool. If there is no available connection object and the maximum limit of the pool size is reached, then it waits for a connection object become available.

The adapter supports top-level business objects as the ones that map to OneWorld interface classes or XML List business objects. It also supports business objects that map to business functions as top-level business objects. The adapter handles the business object based upon the type and structure of the business object. Instead of relying completely on the business objects generated by the ODA, you can create your own hierarchy using the business objects that map to business functions to give a logical representation. For example, the business functions for creating an Order and creating an Order Item can be modeled as a hierarchy. The business objects that map to business functions that create OrderItems would become child business objects of the business object that maps to create the Order business object.

The adapter executes top-level business objects that map to business functions. If the business object does not have any child business objects, the adapter executes the business function that corresponds to the business object. If the hierarchy of such business objects is an input to the business object handler, then, the adapter executes all the business functions in one transaction. In this case, the verb ASI is blank and the flow of the business functions are determined by the order of attributes in the top-level business object. For example, if the top-level business object maps to B110031 and has children B110032 and B110033, then the order of the execution is B110031, B110032, B110033.

If the type attribute of the business object is XMLList, then the business object handler prepares an XML document with values and format defined by the business object. The adapter sends the XML document to OneWorld, using the XMLRequest object. The adapter receives the response as an XML response document and the handler uses the response data from the XML response document to populate the business object.

In some cases, a single simple attribute or an object may need to be used multiple times in a call sequence. You can use attribute ASI to link two attributes. If an attribute is marked as a foreign key, it must have an attribute ASI, `use_attribute_value=` tag, the value of which must correspond to `BusinessObject.AttributeName`. Use this link only if the source business object is of single cardinality. If it is configured with a source business object with multiple cardinality, then the adapter picks up the first business object from the list and maps the value from that business object.

Application event processing

Event notification requires the installation of the event package, `BIA_EVENT`, shipped with the adapter and the creation of event and archive tables in the JDE database.

For information about how to install and configure the `BIA_EVENT` event package, refer to "Installing and configuring IBM event store" on page 19.

The creation, update, or deletion of any record in the JD Edwards OneWorld application can be treated as an event. You can use table triggers, supported by OneWorld, to populate the event table. You can also use any other JD Edwards recommended method to generate events into the event table. During a call to `pollForEvents`, these records are obtained and processed. The event table stores information about the event, as described in Table 6 on page 41.

Note: The Event ID must be unique in the Event table.

Note: The connector uses the information in Table 6 during event subscription to build corresponding business objects and to send those objects to the connector framework for further processing.

Retrieving business objects for event processing

Retrieval of objects for event processing is based on both key and non-key attributes. It is mandatory that the business object support the `Execute` verb if the business object represents a JD Edwards business function and that is supports the `Retrieve` verb if the business objects represents an interface.

Event management

The connector polls the IBM Event table (F5501005) at a regular interval, retrieves the events, and processes the events first by priority and then sequentially. When the connector has processed an event, the event's status is updated appropriately.

The setting of the `ArchiveProcessed` property determines whether the connector archives an event into the IBM Archive table (F5501006) after updating its status. For more information on the `ArchiveProcessed` property, see "Connector-specific properties" on page 17. Table 3 on page 17 illustrates the archiving behavior depending on the setting of the `ArchiveProcessed` property.

How the connector works

This section describes how the different parts of the connector process a business object:

1. Upon startup of the connector, the connector's Agent class performs the following initialization (`Init`) processes:
   - Retrieves configuration properties.
   - Fetches the `Username` and `Password`, and `Environment` from the connector configuration file.

- Creates a OneWorld connector object.
- Logs in to the OneWorld server using the Login method and parameters using the Username and Password as fetched above. This method returns a SessionID.
- Creates an instance of the OneWorld interface object.
- Adds the connector, OneWorldInterface, and SessionID to the connection pool.

2. The OneWorld business object handler reads the verb ASI and translates it into a sequence of callable functions or child objects.
   - If the business object has a child business object of type ACCESS_LEVEL and the Username attribute within this child business object is populated and has a value that is different from what is used by the adapter, then the business object handler opens a new connection using the values of the Username and Password attributes specified for the ACCESS_LEVEL business object. All such business objects must have both Username and Password attributes populated.
   - If the connection creation fails because the application is down, the business object handler returns APPRESPONSETIMEOUT.
   - If the connection creation fails because the Username/Password is wrong, then the business object handler logs an error and returns a FAIL status.
   - If the business object does not have a child business object of type ACCESS_LEVEL, or the value for the Username attribute in this business object is null or has the same value as specified for the adapter Username, then it fetches a connection form the available connection pool. The following steps represent what would happen in the connection pool when the business object handler requests an available connection:
     a. The business object handler checks to see if there are available connections in the pool.
     b. If yes, it checks for the validity of the connection. If it is not valid it attempts to recreate the connection.
     c. If the connection creation fails, it returns APPRESPONSETIMEOUT status.
     d. The business object handler removes the connection from the available list and adds it to the busy list.
     e. If the connection is not available and the maximum number of connections is less than the pool size, then it opens a new connection and adds it to the connection pool's busy list. If opening a new connection returns a failure, the adapter returns APPRESPONSETIMEOUT.
     f. If no connections are available and the maximum limit of the pool size has been reached, then the doVerbFor thread waits until a connection becomes available.
   - If the business object is of type BFN, then the adapter performs the following actions:
     a. The adapter starts a transaction using the BeginTransaction method of the OneWorld class OneWorldInterface.
     b. If the business object maps to an interface class and if the verb ASI is blank, the adapter finds the first method attribute or the first child object that is populated in the business object and executes. it.
     c. If the verb ASI is populated, the adapter calls InvokeMethods, which loops through all the methods specified in the verb ASI.
     d. If the business object maps to a business function, the invoker executes the business function that maps to the business object. If there are child business objects that are not of type ACCESS_LEVEL, the business object

handler loops through them and executes the business functions corresponding to them in the order in which they are defined in the top-level business object.

e. The invoker constructs the arguments based on the attributes defined in the business object and then invokes the method on OneWorld Java objects using reflection APIs.

f. If the execution of the complete business object succeeds, the business object handler commits the transaction using the `Commit` method on object `OneWOrldInterface` and returns a `VALCHANGED` status.

- If the business object is of type `XMLList`, then the adapter performs the following actions:

  a. The adapter creates an XML document with the values and format as specified in the business object.

  b. The adapter sends the document to OneWorld using XML List APIs.

  c. In the event of a failure, the adapter logs an error code and reason in the response document. It also logs errors in the log file with the return status of `FAIL` if there is a problem with the request document.

  d. When the adapter sends the XML document to OneWorld successfully, the values from the response document are updated in the business object.

  e. If the business object has child business objects that are not of type `ACCESS_LEVEL`, the handler repeats the above steps for each child business object.

  f. If the adapter processes the entire business object successfully, the status is set to `VALCHANGED`.

- Releases the connection to the connection pool.
- Returns `VALCHANGED` upon successful execution of the business functions.
- Returns `FAIL` if the business object is of type `BFN` and maps to the `Interface` class and the verb ASI is blank and no attributes are populated.
- Returns `FAIL` if processing fails.

3. The `ConnectionEventStore` class performs the following for subscription delivery:
   - When the connector encounters an event, it
     - creates a business object of the type specified by the event,
     - sets the key and non-key values for the business object (using the object key specified in the event table),
     - sets the verb as `Execute` if the business object is of type business function,
     - sets the verb as `Retrieve` if the business object is of type interface.
   - After it retrieves the business object, the connector sends it to the integration broker with the verb specified in the event.
4. Terminates (`Terminate`) by closing all the connections from the connection pool.

Chapter 2. Installing the adapter

Compatibility

The adapter framework that an adapter uses must be compatible with the version of the integration broker (or brokers) with which the adapter is communicating. The 2.0.4 version of the adapter for OneWorld is supported on the following adapter framework and integration brokers:

- **Adapter framework**:
  - WebSphere Business Integration Adapter Framework versions 2.1, 2.2, 2.3.x, 2.4, and 2.6
- **Integration brokers**:
  - WebSphere InterChange Server, version 4.1x, 4.2.x
  - WebSphere MQ Integrator, version 2.1.0
  - WebSphere MQ Integrator Broker, version 2.1.0
  - WebSphere Business Integration Message Broker, version 5.0
  - WebSphere Application Server Enterprise, version 5.0.2, with WebSphere Studio Application Developer Integration Edition, version 5.0.1

See *Release Notes* for any exceptions.

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

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

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

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

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

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

Assumptions and third-party dependencies

Before you install the connector for JD Edwards OneWorld, review the platform requirements in this section and see the EnterpriseOne documentation for any additional software dependencies, including those specific to your version of EnterpriseOne.

Platform requirements

The connector runs on the following platforms:

- Windows XP
- Windows 2000
- Solaris 8.0
- HP/UX 11i
- AIX 5.2

Installing the adapter for JD Edwards OneWorld and related files

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

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

WebSphere Business Integration Adapter directories and files

After the installation is complete, you can view the file system and its contents. The folders and files created vary depending on the choices made during the installation and on the operating system.

The Installer copies the standard files associated with the connector into your system. It installs the connector agent into the *ProductDir*\connectors\OneWorld directory, and adds a shortcut for the connector agent to the Start menu. Note that *ProductDir* represents the directory where the IBM WebSphere Business Integration Adapters product is installed. The environment variable contains the *ProductDir* directory path, which is *IBMWebSphereAdapters* by default.

Environment variables

If you chose WebSphere MQ Integrator Broker or WebSphere Application Server as your broker, Installer takes the actions described in Table 1 to create and update environment variables on the computer. These actions are not taken if you chose WebSphere InterChange Server as your integration broker, because the environment variables required for that broker are created during installation of the broker.

Table 1. Actions taken by Installer for environment variables

Environment variable name	Installer action
CROSSWORLDS	Creates this environment variable to reference the WebSphere Business Integration Adapter product directory, as specified when using Installer.
MQ_LIB	Creates this environment variable to contain the path to the Java\lib directory within the WebSphere MQ installation, as specified when using Installer.
CLASSPATH	Adds the following entries: *ProductDir*\lib\rt.jar; *ProductDir*\DataHandlers\CwDataHandler.jar;

Table 1. Actions taken by Installer for environment variables (continued)

Environment variable name	Installer action
PATH	Adds the following entries: *ProductDir*\bin\hotspot; *ProductDir*\bin\classic; *ProductDir*\bin;

Connector file structure

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

The utility installs the connector into the *ProductDir*\connectors\OneWorld directory, and adds a shortcut for the connector to the Start menu. Note that *ProductDir* represents the directory where the product is installed.

Table 2 describes the file structure used by the connector, and shows the files that are automatically installed when you choose to install the connector through the Installer.

Table 2. File structure for the connector

Subdirectory of *ProductDir*	Description
\connectors\OneWorld\BIA_OneWorld.jar	Contains classes used by the OneWorld connector only
\connectors\OneWorld\start_OneWorld.bat	The startup script for the generic connector (NT/2000) If you are running the adapter with WebSphere Business Integration Adapter Framework 2.6, you must modify this file. See "Modify startup script for Adapter Framework 2.6" on page 15
\connectors\OneWorld\start_OneWorld.sh	The startup script for the generic connector (Unix). If you are running the adapter with WebSphere Business Integration Adapter Framework 2.6, you must modify this file. See "Modify startup script for Adapter Framework 2.6" on page 15
\connectors\OneWorld\dependencies\BIA_IBMEvents.cmd	Contains the IBM eventstore business function script file. This file can be used when executing the GenJava process to create the Java wrappers of Event Store business functions.
\connectors\OneWorld\dependencies\BIA_EVENT.exe	Executable that installs the eventstore package
\connectors\messages\BIA_OneWorldAdapter.txt	Message file for the connector
\ODA\OneWorld\BIA_OneWorldODA.jar	The OneWorld ODA
\ODA\OneWorld\start_OneWorldODA.bat	The ODA startup file. If you are running the adapter with WebSphere Business Integration Adapter Framework 2.6, you must modify this file. See "Modify startup script for Adapter Framework 2.6" on page 15
\ODA\OneWorld\BIA_OneWorldODA.sh	(Unix users only) The ODA start up file
\ODA\messages\BIA_OneWorldODAAgent_de_DE.txt	Message file for the ODA (German text strings)
\ODA\messages\BIA_OneWorldODAAgent_en_US.txt	Message file for the ODA (US English text strings)
\ODA\messages\BIA_OneWorldODAAgent_es_ES.txt	Message file for the ODA (Spanish text strings)
\ODA\messages\BIA_OneWorldODAAgent_fr_FR.txt	Message file for the ODA (French text strings)
\ODA\messages\BIA_OneWorldODAAgent_it_IT.txt	Message file for the ODA (Italian text strings)
\ODA\messages\BIA_OneWorldODAAgent_ja_JP.txt	Message file for the ODA (Japanese text strings)
\ODA\messages\BIA_OneWorldODAAgent_ko_KR.txt	Message file for the ODA (Korean text strings)
\ODA\messages\BIA_OneWorldODAAgent_pt_BR.txt	Message file for the ODA (Portuguese - Brazil text strings)
\ODA\messages\BIA_OneWorldODAAgent_zh_CN.txt	Message file for the ODA (Simplified Chinese text strings)
\ODA\messages\BIA_OneWorldODAAgent_zh_TW.txt	Message file for the ODA (Traditional Chinese text strings)
repository\OneWorld\BIA_CN_OneWorld.txt	Repository definition for the connector. The default name is BIA_OneWorld.txt.

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

Post-installation tasks

After you have successfully installed the Adapter for JD Edwards OneWorld, complete the following post-installation tasks:

- "Configure the adapter"
- "Copy files"
- "Create an ODBC connection" on page 15

Configure the adapter

After you install the adapter and before you start it for the first time, you must configure the adapter. For details, see Chapter 3, "Configuring the connector," on page 17.

Copy files

During installation, GenJava generates .jar files containing business function classes that are used by the adapter. Copy these .jar files into the `ProductDir\connectors\OneWorld\repository` folder. The adapter and ODA dynamically upload these files.

To prepare the Java wrapper for business functions used by the event notification component, run GenJava using the BIA_IBMEvents.cmd file that comes with the adapter. Copy the generated .jar file to the `ProductDir\connectors\OneWorld\dependencies` folder.

Note: Since the dependency files may be version specific, see the OneWorld orEnterpriseOne documentation before preforming this task.

OneWorld 8.0

Copy the following files, present in the `B7334\System\classes` folder to the `ProductDir\connectors\OneWorld\dependencies` folder:

- `Kernal.jar`
- `Connector.jar`

Copy the jdeinterop.ini file from the `B7334\Interoperability\Examples\Java` folder to the `ProductDir\connectors\OneWorld\dependencies`. For more information on editing the jdeinterop.ini file, refer to the *Interoperability Guide*.

EnterpriseOne 8.9.x

Copy the following files, present in the `B9\System\classes` folder to the `ProductDir\connectors\OneWorld\dependencies` folder:

- `Kernal.jar`
- `Connector.jar`
- `Logic.jar`
- `Jdeutil.jar`
- `Database.jar`

Note: As JD Edwards updates its releases, these dependencies may change. Refer to JD Edward's PeopleSoft EnterpriseOne Tools 8.9.x PeopleBook: Connectors-> Understanding the Java Connector->Installing a Java connector.

Copy the sample jdelog.properties and jdeinterop.ini.templ file from the `B9\System\classes\Samples` folder as jdeinterop.ini file in the `ProductDir\connectors\OneWorld\dependencies` folder.

Edit the jdelog.properties and jdeinterop.ini files. For information on editing sections of the filter, refer to the *OneWorld Interoperability Guide*.

Create an ODBC connection

The OneWorld adapter requires an ODBC data source for each DB2 UDB database in order to start and pull the connector. For information about how to create an ODBC connection, refer to the JD Edwards *Installation Guide*.

Modify startup script for Windows 2000

If you are running the adapter on Windows 2000 and using InterChange Server as a broker, modify the startup script to avoid an invocation error. The error occurs when you start the adapter. The invocation fails because the startup command is too long. Modify the startup script by replacing each instance of `"%CONNDIR%"` with `".."`, the relative path symbol.

For example, modify `%CONNDIR%\repository` to `..\repository`

For more on the script changes, see ftp://ftp.software.ibm.com/software/ts/cw/adapters/AdapterForOneWorld/WIN/OneWorld_Adapter_Script_Ch

Modify startup script for Adapter Framework 2.6

If you are running the Adapter for JD Edwards OneWorld with WebSphere Business Integration Adapter Framework 2.6, you must modify startup scripts after installing, but before starting, the connector. Otherwise, the Adapter for JDE OneWorld may not work with the new framework. For more information and sample startup scripts, see “Startup scripts for Adapter Framework 2.6” on page 97.

Chapter 3. Configuring the connector

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

Connectors have two types of configuration properties: standard and adapter-specific. You must set the values of these properties using Connector Configurator before running the adapter. For further information, see "Startup scripts for Adapter Framework 2.6" on page 97.

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

Standard connector properties

Standard connector configuration properties provide information that all adapters use. See Appendix A, "Standard configuration properties for connectors," on page 55 for documentation of these properties.

Connector-specific properties

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

To configure connector-specific properties, use Connector Configurator. Click the **Application Config Properties** tab to add or modify configuration properties. For more information, see "Startup scripts for Adapter Framework 2.6" on page 97.

Table 3 lists the connector-specific configuration properties for the connector, along with their descriptions and possible values. See the sections that follow for details about the properties.

Table 3. Connector-specific configuration properties

Name	Possible values	Default value
Username	JDE	None
Password	JDE	None
PoolSize	5	5
Environment	DV7334	None
ServerName	JDEDEV1	None
PortNo	6010	None

Table 3. Connector-specific configuration properties (continued)

Name	Possible values	Default value
PollQuantity	Any number.	1
EventStoreFactory	`com.ibm.adapters.oneworld.OneWorldEventStoreFactoryInstance`	`com.ibm.adapters.oneworld.OneWorldEventStoreFactoryInstance`
InDoubtEvents	Reprocess, Fail on startup, Log error, Ignore	`Ignore`
ArchiveProcessed	True, False	`True`
UseDefaults	True	True

Username

The Username for connecting with the OneWorld application. This is a required property.

Password

The Password for connecting with the OneWorld application. This is a required property.

PoolSize

The maximum number of connections in the Connection Pool. This should not be more than the maximum allowed connections specified in the `JDEInterop.ini` file. This is a required property.

Environment

Name of the environment in which the connection has to be done within OneWorld. This is a required property.

ServerName

Name of the machine on which the OneWorld server is running. This is a required property when using XML List business objects.

PortNo

The port number on which the OneWorld server is listening. This is a required property when using XML List business objects.

PollQuantity

The number of events to be fetched from the application for each poll cycle.

EventStoreFactory

This property is the name of the event store factory instance class. The value is `com.ibm.adapters.omeworld.OneWorldEventStoreFactoryInstance`.

InDoubtEvents

Decides whether or not to reprocess incomplete events. Default value is `Ignore`

ArchiveProcessed

Checks if the default value is set or not. If `ArchiveProcessed` is set to `true`, the archival business function archives the event into the archive table. If the property is set to `false` or is not set, the archival business function is not called. The default value is `true`.

UseDefaults

Checks if the default values are set. The default value is `true`. This property is not used by the adapter.

Starting the connector

To start the connector, run the `start_OneWorld.bat` or `start_OneWorld.sh` script.

Note: If you are running the adapter with WebSphere Business Integration Framework 2.6, you must modify the start script. For more information, see "Modify startup script for Adapter Framework 2.6" on page 15.

If the connector is running with WebSphere MQ Integrator Broker, you must also set the `-c` option.

The start command for the connector has this format:

```
<script name> <connector name> {<ics name> | -c <MQI configuration file>}
```

So, for WebSphere InterChange Server:

```
start_OneWorld OneWorld WebSphereICS
```

Or, for WebSphere MQ Integrator Broker:

```
start_OneWorld OneWorld -c C:\WebSphereAdapters\connectors\OneWorldAgentConfig.cfg
```

In Windows, select **Start>Programs>IBM WebSphere Business Integration Adapters>Connectors>OneWorld** to start the connector for OneWorld.

Stopping the connector

To stop the connector, you can use Adapter Monitor. The toolbar lets you activate, de-activate, pause, shutdown and delete the connector.

Installing and configuring IBM event store

The adapter package includes an executable `BIA_EVENT.exe` file. This executable file installs the IBM event store components. JD Edwards refers to this as a software update. Follow the JD Edwards instructions for applying a software update. Event store (software update) components comprise an event and archive table and business functions used for retrieving, deleting, updating, and archiving events in the event and archive table.

Note: For JD Edwards OneWorld version 8.93, access updated event store component files `WBIAEVTPK.exe` and `WBIAEVTPK.cab` at the following locations:

ftp://ftp.software.ibm.com/software/ts/cw/adapters
/AdapterForOneWorld/WIN/WBIAEVTPK.exe

ftp://ftp.software.ibm.com/software/ts/cw/adapters
/AdapterForOneWorld/WIN/WBIAEVTPK.cab

Then use these files to install the event store. Both files include the full software master and its directories. If you cannot download the `.exe` file due to size restrictions, download the smaller `.cab` file and decompress it before installing.

Business functions for event handling, table definition files, and data items are part of the event package `BIA_EVENT.exe`. You must prepare the deployment server or workstation before executing a software update, then follow the software update procedure. The JD Edwards *Software Update Installation Guide* describes the preparation and update procedures in detail.

The `BIA_EVENT.exe` file creates the required business functions and table definition scripts, however you must create the event and archive tables using the JD Edwards client. Once you have successfully installed the software update on the deployment server or workstation you must deploy the components to the enterprise server so the adapter can find the event store.

The contents of the `BIA_EVENT` package are as follows:

- `B551005` — `Retrieve_WBIAEvents`
- `B551006` — `Update_WBIAEvent`
- `B551007` — `Archive_WBIAEvent`
- `B551008` — `Delete_WBIAEvent`
- `B551009` — `Recover_WBIAEvent`

The following are the Tables Definitions files:

- `F5501005.h` — Event table structure
- `F5501006.h` — Archive table structure

The following data items are included:

- `EVENT_ID`
- `EVT_DESC`
- `EVT_PRTY`
- `EVT_STATUS`
- `EVT_TIME`
- `ADAPTER_ID`
- `ARCHIVE_T`
- `OBJ_KEY`
- `OBJ_NAME`
- `OBJ_VERB`

Note: Other data items exist in the package but are not used by the tables and will be removed from the package in the next release.

Population of an event into the event table

To populate an event to the event table, you can either use OneWorld table triggers or any other JD Edwards recommended method. The event business object structure is as follows:

- **Object Key**:
  - A required field
  - The unique identifier for the business object row for which the event was created
  - The format of the value should corresponded to the business object definition generated by the ODA in ways such as ChildBOName.keyAttribute=123 (where ChildBOName is of type Business Function)
  - Values provided in this field are case sensitive and should match the business object definition
  - The maximum key value (when multiple attributes constitute a key) is 250 characters
- **Object Name**:
  - A required field
  - The name of the OneWorld business object for which the event is detected
- **Object Verb**:
  - A required field
  - The verb for the event
  - Any verb that is supported at the business object level
  - When InterChange Server is the broker, the verb at the child business object level must be the same as the top-level business object. For example, if the top-level business object supports the verbs Create and Retrieve, all child business objects should support the same Create and Retrieve verbs.
- **Priority**:
  - Event priority is defined as an integer value in the range 0 - n, with 0 having the highest priority
  - The adapter polls and processes events in order of priority
  - **Status**:
    - The event status is initially set to 0, which implies a status READY_FOR_POLL
- **Description**: Any comment associated with the event
- **Event ID**: A unique ID associated with the event row
- **ConnectorId**: Identifies the connector in a multiple connector configuration
- **Event Ts**: Event creation time stamp.

Using log and trace files

The adapter components provide several levels of message logging and tracing. The connector uses the adapter framework to log error, informational, and trace messages. Error and informational messages are recorded in the log file, and trace messages and their corresponding trace levels (0 to 5) are recorded in a trace file. For details about logging and trace levels, see Chapter 6, “Error handling and event codes,” on page 49.

You configure both the log and trace file names, as well as the trace level, in Connector Configurator. For details about this tool, see “Startup scripts for Adapter Framework 2.6” on page 97.

Error messages for ODA are logged into the trace file name specified for the ODA. If an incorrect trace file name is specified, then the messages are sent to the screen prompt where the ODA is running. Trace files and the trace level are configured in Business Object Designer. The process is described in “Configuring the agent” on page 24. The ODA trace levels are the same as the connector trace levels, defined in “Tracing” on page 51.

Chapter 4. Creating and modifying business objects

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

This chapter contains the following sections:

Overview of the ODA for OneWorld

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

The Object Discovery Agent (ODA) for JD Edwards OneWorld generates business object definitions from the .jar files generated by GenJava. The Business Object Wizard automates the process of creating these definitions. You use the ODA to create business objects and Connector Configurator to configure the connector to support them. For information about Connector Configurator, see "Startup scripts for Adapter Framework 2.6" on page 97.

Generating business object definitions

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

Starting the ODA

The ODA can be run from any machine that can mount the file system on which the metadata repository (that is, the IDL files) resides, using the `start_OneWorldODA.bat` (NT/Windows 2000) or `start_OneWorldODA.sh` (Unix) start file. This file contains start parameters, including the paths to certain required OneWorld and connector .jar files.

Note: If you are running the adapter with WebSphere Business Integration Adapter Framework 2.6, you may need to modify the ODA start script. For more information, see "Modify startup script for Adapter Framework 2.6" on page 15.

The ODA for OneWorld has a default name of `OneWorldODA`. The name can be changed by changing the value of the `AGENTNAME` variable in the start script (`start_OneWorldODA.bat`).

To start the ODA, run this command:

```
start_OneWorldODA
```

Running Business Object Designer

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

- "Selecting the agent"
- "Configuring the agent"
- "Selecting a business object" on page 26
- "Confirming object selection" on page 27

Selecting the agent

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

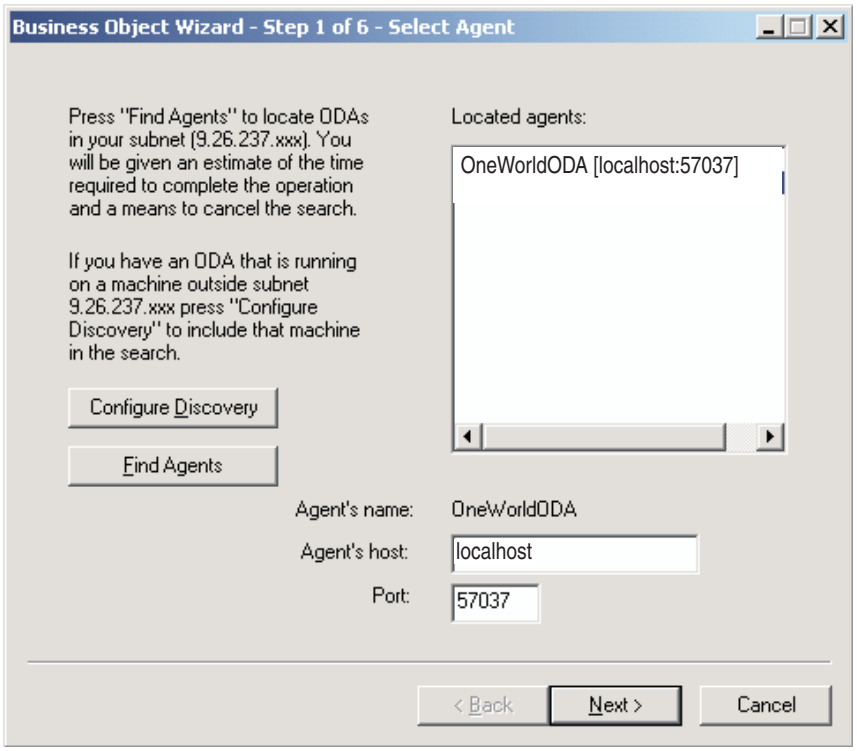

Figure 5. Select Agent screen

Configuring the agent

After you click **Next**, the *Business Object Wizard - Step 2 of 6 - Configure Agent* screen appears. Figure 6 on page 25 illustrates this screen with sample values.

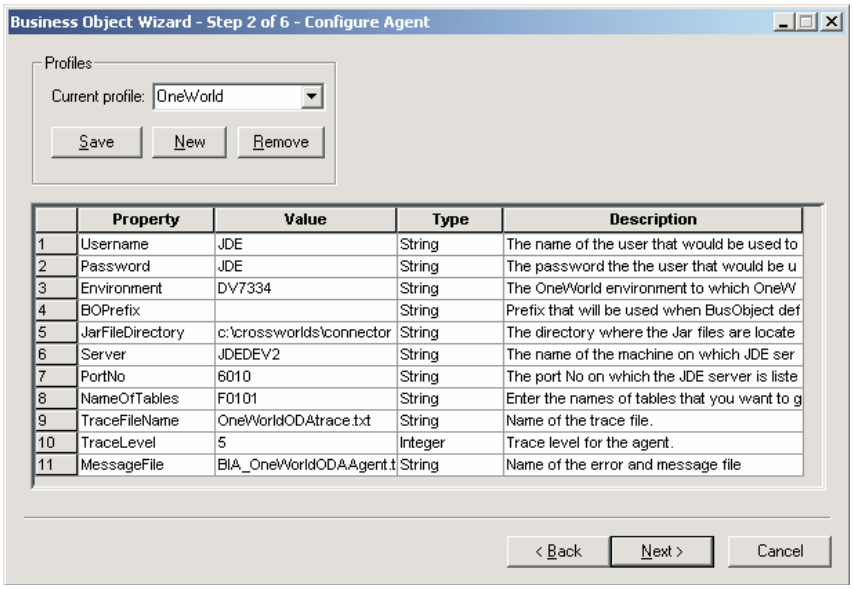

Figure 6. Configure Agent screen

The properties you set on this screen are described in Table 4. You can save all the values you enter on this screen to a profile. Instead of retyping the property data next time you run the ODA, you simply select a profile from the drop-down menu and re-use the saved values. You can save multiple profiles, each with a different set of specified values.

Table 4. Configure Agent properties

Property name	Default value	Type	Description
`BOPrefix`	None	String	The prefix that the ODA prepends to the names of the business objects it generates.
`Environment`	None	String	The environment that is used to connect to OneWorld.
`JarFileDirectory`	None	String	(Required) The directory where the .jar files are located. All the .jar files having business functions that must be invoked using the adapter must be placed in this directory.
`NameOfTables`	None	String	The names of the tables in OneWorld for which business objects have to be generated delimited by `;`, for example, `F4211;f4210`.
`Password`	None	String	The password that is used to connect to OneWorld.
`PortNo`	None	String	The port number on which the JDE server is running.
`Server`	None	String	The machine name on which JDE is running.
`TraceFileName`	None	String	The name of the trace message file; for example, `OneWorldODAtrace.txt`.

Table 4. Configure Agent properties (continued)

Property name	Default value	Type	Description
TraceLevel	5	Integer	(Required) The tracing level (from 0 to 5) for the Agent. For details about tracing levels, see "Tracing" on page 51.
MessageFile	None	String	(required) The name of the message file that contains all the messages displayed by the ODA. For OneWorld, the name of this file is BIA_OneWorldODAAgent.txt. If you do not correctly specify the name of the message file, the ODA will generate an error.
UserName	None	String	The username that is used to connect to OneWorld.

1. Use the **New** and **Save** buttons in the Profiles group box any time you want the ODA to create a new profile. When you use the ODA again, you can select an existing profile.
2. Type the value of each property, as defined in Table 4 on page 25.

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

Selecting a business object

The *Business Object Wizard - Step 3 of 6 - Select Source* screen appears, as illustrated in Figure 7 on page 27.

The following lists the rules associated with selecting objects for generation:

- Selecting a parent object automatically selects the children objects for generation. If you select the parent object as well as the children, then only the children object that you select are generated.
- Selecting a child object without selecting its parent causes the child object, but not the parent, to be generated.
- All child business objects are generated with single cardinality containment relationships.
- If a business object requires multiple cardinality behavior, then you must manually change the cardinality using Business Object Designer.
- The ODA does not mark the key fields of the business objects so you must manually mark the key fields in the business object before saving.
- You can select a .jar file for generation. This generates the definition of all the interface and business function business objects contained in the .jar file.

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

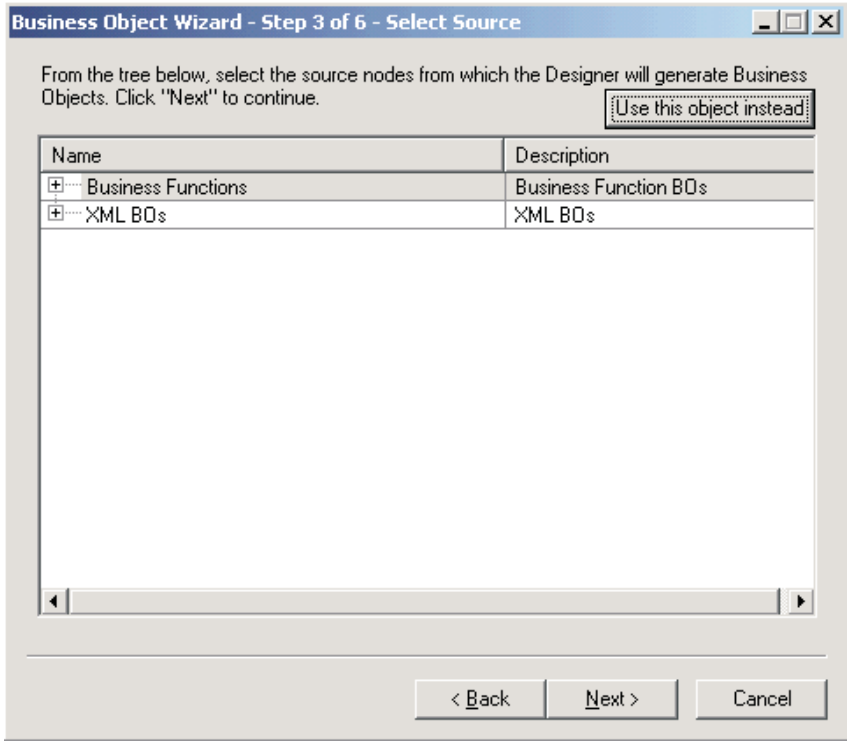

Figure 7. Select Source screen

The ODA displays two options as tree nodes, `Business Functions` and `XML BOs`. Clicking `Business Functions` displays the tree containing the OneWorld objects in the .jar files found under the directory specified against the `JarFileDirectory` configuration property. For information specific to `Business Functions`, see "Source node selection" on page 28

Clicking `XML BOs` displays the child tree nodes. These child tree nodes correspond to the names of the tables that were entered in the ODA properties window. From the tables listed, you can select which tables are generated. For information specific to `XML BOs`, see "XML list business objects" on page 31

You can choose objects from multiple .jar files for business object generation. To select the objects, use the `Use This Object Instead` button. The standard filter feature allows you to select a subset of child nodes of a tree.

1. If necessary, expand a OneWorld module to see a list of sub-objects.
2. Select the OneWorld objects that you want to use.
3. Click **Next**.

Confirming object selection

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

For business objects generated by using the ODA, key fields must be manually marked in Business Object Designer before saving the business objects. The ODA does not mark any attributes as key fields. If you plan to map values from one business object to another business object, you must mark the foreign keys. When

the business object attribute needs a value from some other business object that has already been processed, the ASI tag, `use_attribute_value`, must be manually added to the attribute ASI.

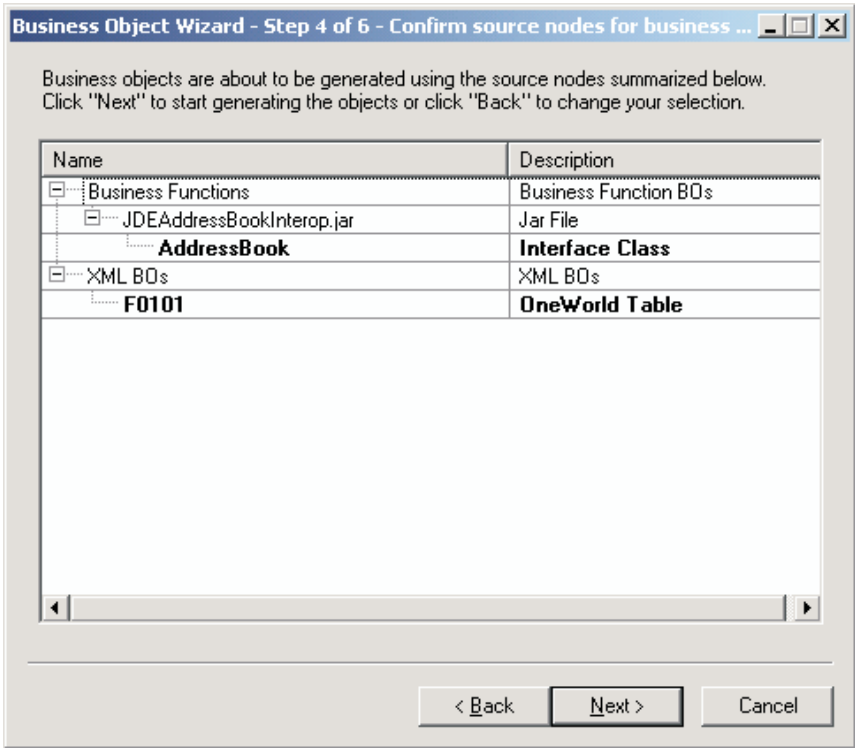

Figure 8. Confirm source node screen

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

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

Source node selection

You can generate both business function business objects or XML List business objects in one run of the ODA. Parallel generation of both kinds of business objects is supported.

Business functions: If the business function business objects are being generated, the ODA will generate business objects for the OneWorld objects that you select by loading and introspecting all the selected classes. ODA uses the `BOProperties` window to get the user's configuration information for each OneWorld object.

There are two types of business object properties windows for business objects that map to interface classes. The first window, `Capturing` supports verbs to capture the supported verbs for all selected OneWorld objects. The second window, `Capturing Method sequence for Verb ASI`, captures the method sequence for each verb of each business object. For data structure business objects, one default verb is created, `Execute`, and this verb has no ASI.

OneWorld objects can have attributes and methods. The names of the business object definitions are derived from the OneWorld object names and appended with

the business object prefix. If there are two business functions that use the same data structure class, the ODA generates multiple definitions with the same structure with the exception of a business function name in the business object ASI. The names of the business objects are incremental, starting with _1, _2... and so on. For example, if D0100033 is used by two functions, the business objects generated are D0100033, D0100033_1.

Get/Set Methods

For the OneWorld data structure object, the get/set methods have corresponding business object attributes. The type of the attribute is stored in the ASI as type=<type> and the actual name of the attribute is stored in ASI as name=<name>. For a combination of get/set, only one attribute is generated. For example, if an attribute name is ID, the methods are getID() and setID(). In this case, the business object has one attribute with the name ID and ASI as getter=getID();setter=setID(), where type=int, and name=ID.

Business function attributes

An attribute is created for each business function defined in the OneWorld interface java class. For proper representation of business function call, the type of this attribute will be a child business object containing attributes representing input parameters of the business function call. The name of the attribute would be the name of the business function and the type would be the name of the data structure business object. The attribute has an ASI that holds the name of the business function using the tag bfn_name=.

Business function parameters

The input parameters of a business function are represented as attributes. If the data type of a particular parameter is something that is unsupported by the WebSphere Business Integration format, they are represented as 'String'. Basically, the parameters are the variables defined in the data structure class. The ASI of a parameter stores the original data type value and name. For a list of supported parameters, see "Business object attribute types" on page 39.

Object types in OneWorld

There are two types of java classes present in OneWorld jars generated using GenJava:

- Interface class file

  The interface class files map to the library and an interface combination is defined in the iJDEScript file. The file has the signature for all the business functions as well as create<businessfunction>ParameterSet methods for the business functions that are imported in a specified interface.

  If the file is as shown below:

```
login
library JDEAddressBook
   interface AddressBook
      import B0100031
      import B0100019
      import B0100032
      import B0100002
      import B0100033
build
logout
```

The class file for `AddressBook.class` would have method for B0100031 as well as Create<B0100031>ParameterSet. Where <B0100031> is the English text name of the business function.

- Data structure class file

  The data structure class files maintain the get and set methods for all the parameters that are required as input for a specific business function.

Business objects that map to interface classes: You can specify business object information for business objects that map to interface classes. After you create a business object, you can specify the verbs that are valid for the object, the method sequence of a given verb on the object, the business object-level ASI, and the attribute-level ASI. This section describes how to specify this information, using the ODA with Business Object Designer. For a detailed description of these categories of information and what they mean for business object structure in the JD Edwards OneWorld connector, see Chapter 5, "Understanding business objects," on page 37.

Selecting verbs

In Business Object Designer, if a selected business object maps to an interface object in OneWorld, the first screen that appears when you finish creating a business object and open it in a separate window is the *BO Properties - Select Verbs for component* screen. Figure 9 illustrates this screen for the AddressBook business object. For business objects that map the business functions of `XML List`, a single verb `execute` is created.

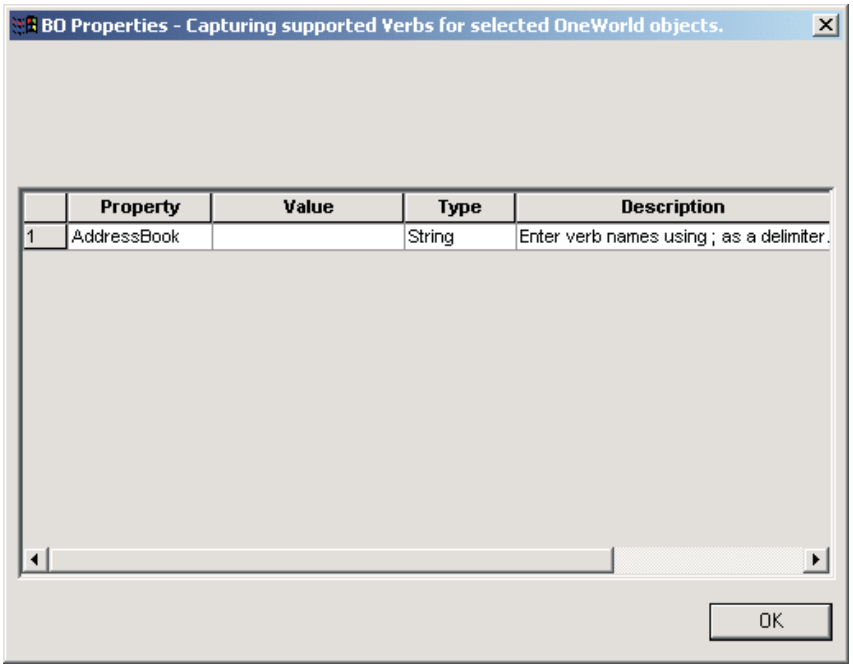

Figure 9. Select verb for component screen

On this screen, you can specify the verbs that the business objects supports. You can specify the verbs that you need for a specific business object by typing the verb names and delimiting them with a **;**. The verb names must follow the naming convention as specified in the *Business Object Development Guide*.

The standard verbs used in WebSphere Business Integration are `Create`, `Retrieve`, `Delete`, and `Update`. For details about business object verb ASI for the OneWorld connector, see "Verb ASI" on page 42.

Specifying the verb ASI

For each verb selected, a separate window appears where you specify the business function sequence that must be executed for the verb.

Figure 10 illustrates this screen for the `Retrieve` verb of the `AddressBook` business object under `Business Functions`, created in Figure 7 on page 27 and Figure 8 on page 28.

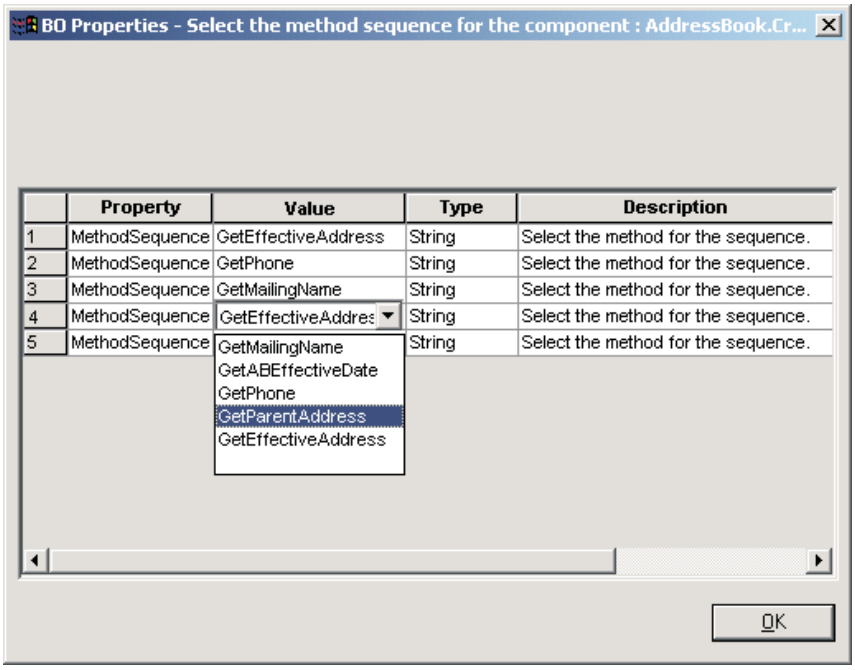

Figure 10. Setting the verb method sequence

From the **Value** list for the `MethodSequence` property, you can select the method that you want the business object to execute first for the verb. In Figure 10, the method sequence is as follows:

- The first method that will be executed in the sequence of methods for the `Retrieve` verb is `GetEffectiveAddress`.
- The second method in the sequence is `GetPhone`.
- The third method in the sequence is `GetMailingName`.

By specifying a business function sequence for the verb, you are creating the verb ASI that is associated with that verb. If necessary, this verb ASI can be modified later.

XML list business objects: For each table that you select on the Confirm source node screen, seen Figure 8 on page 28, subsequent properties windows are displayed to capture additional information.

There are three types of business object properties windows displayed for XML business objects:

- Table type: Captures the table type for all selected OneWorld tables.
- Data selection: Allows you to select the where clause properties for each table.
- Data sequencing: Allows you to select the data sequencing for the query

Table types

The business object properties window displays the property names and property values. You can select the table type for each table. The drop-down menu allows single cardinality selection.

You can select from the following property values:

- OWTABLE
- OWVIEW
- FOREIGN_TABLE

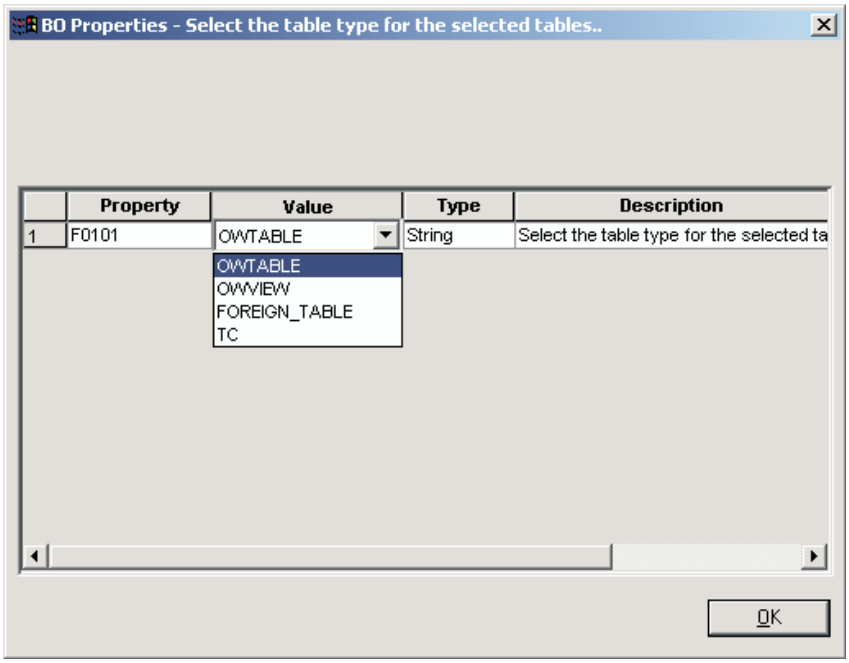

Figure 11. Select the table type for selected tables screen

Once the table type is selected, the ODA invokes the `GetTemplate` API for each table. The ODA prepares the XML document for the `GetTemplate` call and passes it to OneWorld. The response XML document gives the list of columns that are present in the table. The business objects generated have all these columns displayed as attributes in the business object.

Data selection

This business objects properties window, seen in Figure 12 on page 33, captures the where clause properties for the columns of each table. This window is displayed for each table. The property names are the names of columns in the table. The

property values represent the where clause parameters that have to be used for a column. This is populated only for columns that are part of the where clause. The format of the value must be:

```
clause_type=<ClauseType>;
clause_seq=<Clause Sequence>;
operator_type=<Operator Type>;
```

The `<Clause Type>` is either WHERE, AND or OR. The `<Clause Seq>` is a number representing the order in which the columns are added in the where clause. The `<Operator Type>` can be one of the following values:

- EQ (equal to)
- NE (Not equal to)
- LT (Less than)
- GT (greater than)
- LE (less than or equal to)
- GE (greater than or equal to)

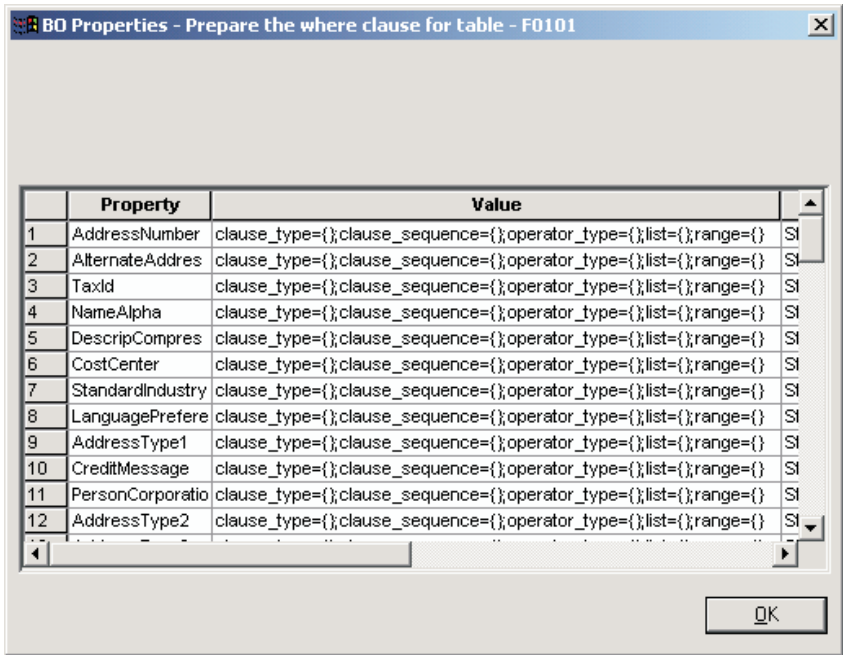

Figure 12. Prepare the where clause for table screen

Data sequencing

The data sequencing screen, seen in Figure 13 on page 34, lists property names and allows you to select either ascending (ASCD) or descending (DSCD) sequencing order for a specific column from a drop-down menu.

Besides the type of sequencing, the adapter uses an ASI tag, `sort_order`, to order the attributes when the ordering clause is prepared. This property is not added by ODA; you must manually add this property to the attribute ASI at the time you select the sorting property.

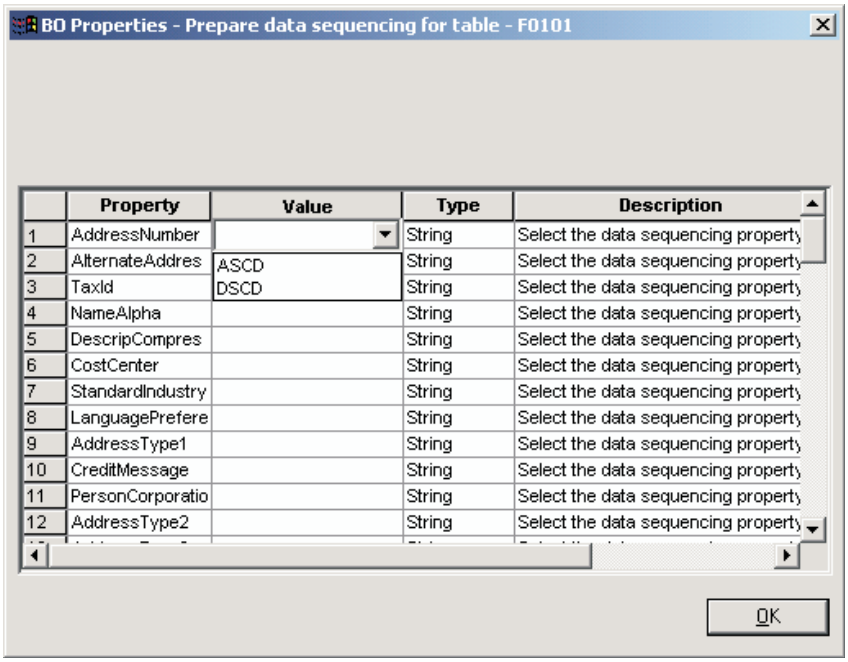

Figure 13. Prepare data sequencing screen

Opening the business objects in a separate window

The *Business Object Wizard - Step 6 of 6 - Save business objects* screen appears with options to save a copy of the business objects to another file, to open the new business objects in another window, and to shut down the OneWorld ODA. If you choose to open the new business objects in another window, the Business Object Designer displays a window where you can modify the attributes for those business objects.

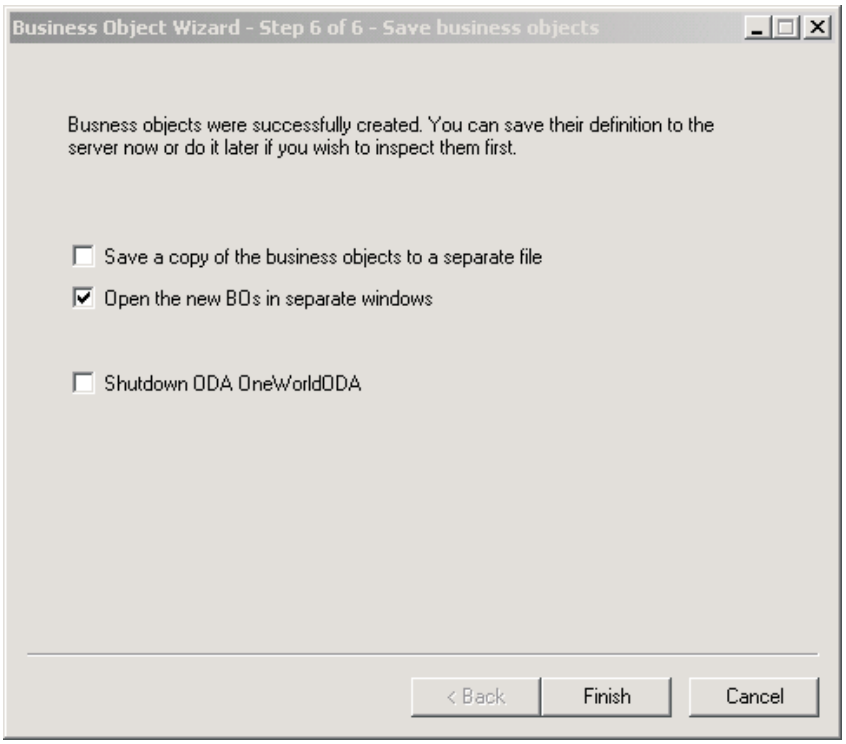

Figure 14. Save business objects screen

To open the business objects in separate window:

1. Select **Open the new BOs in separate windows**. A dialog box appears.
2. Click **Finish**. Each business object appears in a separate window where you can view and set the ASI information for the business objects and business object verbs you just created. For details, see "Business objects that map to interface classes" on page 30.

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

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

Business Object Designer saves the files to the specified location.

If you have finished working with the ODA, you can shut it down by selecting the check-box, `Shutdown ODA JD Edwards OneWorld ODA` before clicking **Finish**.

Uploading business object files

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

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

Chapter 5. Understanding business objects

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

The chapter contains the following sections:

Defining metadata

The connector for JD Edwards OneWorld is metadata-driven. In the WebSphere business integration system, metadata is defined as application-specific information that describes a OneWorld application object's data structures. The metadata is used to construct business object definitions that the connector uses at runtime to build business objects.

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

A metadata-driven connector handles each business object that it supports according to the metadata encoded in the business object definition. This enables the connector to handle new or modified business object definitions without requiring modifications to the code. New objects are created in Business Object Designer, with or without the assistance of the ODA. To modify an existing object, use Business Object Designer directly (the ODA cannot be used to modify an existing business object).

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

The connector makes assumptions about the structure of its supported business objects, the relationships between parent and child business objects, and the format of the data. Therefore, it is important that the structure of the business object exactly mirror the structure defined for the corresponding JD Edwards OneWorld object or the adapter will not be able to process business objects correctly.

If you need to make changes to the business object structure, make them to the corresponding object in OneWorld and then run GenJava to create .jar files that can be used as an input into the ODA.

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

Connector business object structure

Each OneWorld object has a corresponding business object.

The business objects for the JD Edwards OneWorld adapter can be of two types to support the Java APIs as well as the XMLList APIs from OneWorld.

Business function business objects

The business objects that are processed using Java APIs have a `type=BFN` tag in the business object ASI that distinguishes them from XML List business objects. Within the business objects that are used for Java APIs, there are two types of business objects: one type that maps to the `Interface` class and another type that maps to a business function class. The business objects that map to the `Interface` class have ASI as follows:

```
type=BFN;class_name=com.JD Edwards.interop.AddressBook.JDEAddressBook
```

Interface business objects

All the business functions present in the class `JDEAddressBook` and present in the .jar file generated by GenJava are represented as child business objects to the main business object. All the child business objects for a top-level business object must map to business functions that can be access through one username and password. If a specific business function has a different username and password access, then that must be part of a separate business object hierarchy that has access to that username and password. Refer to "Business functions" for further details on how to define special access permissions for a business object.

Each data structure class present in the .jar file, generated from GenJava, maps to the corresponding business object. For example, in the `JDEAddressBook` example, the data structure names are:

```
D0100031
D0100019
D0100032
D0100002
D0100033
```

And, they map to business functions and child business objects created for `B0100031`, `B0100019`, etc.

Interface business object attributes

First Paragraph

The ASI for the interface business objects has a tag for `name=` where the value is the name of the data structure. It also has a tag, `bfn_name=`, for the name of the business function that corresponds to these business objects.

The attribute name of business function interface business objects maps to the name of the method that is represented by the business object. For example, if the data structure is `D0100033`, the name of the attribute in the `AddressBook` business object would be `GetEffectiveAddress`. The ASI at this attribute level would give the name of the method using the ASI tag `bfn_name=`.

Business functions

The OneWorld connector invokes all of the business functions in one `doVerbFor()` call in one transaction. If one of them fails, all of them will be rolled back. All the business functions within one business object execution must have access permissions from a single user. The user can be one that is created for the adapter and is maintained as a connection pool or it can be a specific user. You can specify a user for a business object by using a child business object of single cardinality with the type `ACCESS_LEVEL`.

For proper representation of business function calls, the business function is modeled as a child business object containing attributes that represent data structure variables.

The adapter supports the business objects that map to business functions to be executed independently. For all such business objects, the ASI contains the information required to execute the business function, for example, business function name, and the name for the data structure. The ASI for a business function business object, as above, can be represented as follows:

```
bfn_name=getEffectiveAddress
type=BFN
name=com.JD Edwards.interop.D0100031
```

Business function attributes

For each attribute present in the Data Structure class, a corresponding business object attribute is generated in the business function business object. The ASI for the attribute holds information about its type and name in OneWorld. For example, if the attribute type is `JDEDate`, then the ASI holds `name=EffectiveDate;type=JDEDate`. Besides some simple types of attributes, OneWorld supports two proprietary data types, `JDEDate` and `JDEMathNumeric`.

JDEDate: The following methods are available in this OneWorld Java class:

- `JDEDate()` — constructor
- `GetDay()` — returns the day of the date
- `GetMonth()` — returns the month of the date.
- `GetYear()` — returns the year of the date.
- `SetDay(short)` — set the day of the date
- `SetMonth(short)` — set the month of the date
- `SetYear(short)` — set the year of the date

The values for attributes that map to OneWorld date fields are specified in the format `MM/DD/YYYY`. The adapter parses this string value and calls OneWorld APIs on the `JDEDate` object to set values for the day, month, and year. If the data from OneWorld has to be set in the business object, it uses the `get` methods to set value in the attribute.

JDEMathNumeric: The following methods are present in the `JDEMathNumeric` class:

- `GetValue()` — returns the value as a String, for example, `12345.6789`
- `SetValue()` — sets the value from a String, for example `12345.6789"`

Business object attribute types: The following table lists the data types supported by OneWorld and the corresponding type in a WebSphere Business Integration business object.

Table 5. Business object attribute types

OneWorld type	Business object attribute type	ASI
`JDEDate`	Date	`type=JDEDate`
`JDEMathNumeric`	Integer	`type=JDEMathNumeric`
`int`	Integer	`type=int`
`boolean`	Boolean	`type=boolean`

Table 5. Business object attribute types (continued)

OneWorld type	Business object attribute type	ASI
char	String	type=char
String	String	type=String
short	Integer	type_short
float	Float	type=float
double	Double	type=double
byte	String	type=byte
long	Integer	type=long

XML list business objects

The XML business objects map to the OneWorld table. The business object attributes map to the columns of the table.

If the components have columns with the same name, the ODA generates unique attribute names by appending them with "_" followed by a number. For example, if the field, AddressNumber, appears multiple times, the attributes generated will have names AddressNumber, AddressNumber_1, AddressNumber_2 and so on. The maximum length of the attribute is set, based on what is returned from the getTemplate() API call.

Custom business functions

The following custom business functions are included and required for implementation of event notification in the adapter:

- Retrieve_WBIAEvents — The name of the business function that retrieves the records from the IBM events table.
- Update_WBIAEvents — The name of the business function that updates the records from the IBM events table.
- Delete_WBIAEvents — The name of the business function that deletes the records from the IBM events table.
- Archive_WBIAEvents — The name of the business function that archives the records from the IBM events table to the archive table.
- Recover_WBIAEvents — The name of the business function that retrieves the IN_PROGRESS events and changes the status to READY_FOR_POLL.

Events business object structure

The following table details the event notification features supported by the connector.

Table 6. Event table structure

Columns	Description
OBJ_KEY	The unique identifier that identifies the business object row for which the event was created. If there are multiple attributes in a business object that make a key then the values are name value pairs delimited by ";" If the business object is of type business function, then the object key should be as follows: DS0013keyattr1=123; DS0013keyattr2=124. If the verb ASI for the retrieve verb specifies multiple business functions, you might need to set multiple key fields. That is supported since the adapter is using the data structure name along with the attribute name, for example, D0013.attr1=123;D0012.attr1=345.
OBJ_NAME	OneWorld business object for which the event was detected.
OBJ_VERB	Verb for the event.
EVT_PRIORITY	Event priority.
EVT_STATUS	Event status. Initially set to READY_FOR_POLL.
EVT_DESC	Any comment associated with the event.
EVENT_ID	Unique ID of the event row.
ADAPTER_ID	Identifies the connection in a multiple connector configuration.
EVT_TIME	Event creation timestamp.
ROW_ID	Archive record ID, generated by OneWorld.
PROC_TIME	Event processing timestamp.

Creating, packaging, and deploying custom business functions

For step-by-step instructions on creating custom business functions for event notification, go to: ftp://ftp.software.ibm.com/software/ts/cw/adapters /AdapterForOneWorld/IBM_Event_Package.doc

For step-by-step instructions on packaging and deploying custom business functions for event notification, go to: ftp://ftp.software.ibm.com/software/ts/cw/adapters
/AdapterForOneWorld/Packging_and_Deployment.doc

Application-specific information for business functions

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

Application-specific information can be specified for the overall business object as well as for each business object attribute.

Business object-level ASI

Object-level ASI provides fundamental information about the nature of a business object and the objects it contains. Business object ASI is in name-value pairs. Business objects that represent interface objects recognize the following ASI names:

- type=BFN if the adapter invokes a business function.

- class_name=com.JD Edwards.interop.AddressBook, which is the name of the interface class.

Business objects map to business functions that are executed individually. For business objects that represent business functions, the adapter recognizes the following names:

- type=BFN if the adapter invokes a business function
- bfn_name=getEffectiveAddress
- name=com.JD Edwards.interop.D0100031, which is the name of the structure class.

Verb ASI

For business objects that map to the Interface class, the Verb ASI contains a sequence of attribute names that map to business functions for the OneWorld BO Handler to call. The adapter invokes business functions in the sequence specified by the Verb ASI.

For business objects that map to business functions, the Verb ASI is blank.

Attribute-level ASI

The business objects that map to business functions have attributes that map to the get<Attr>/set<Attr> method combinations of the data structure class. The connector takes this data structure object as an input parameter when the function is invoked. For all such attributes, ASI stores the type of the attribute as type=<type> and the actual name of the attribute as name=<name>. The adapter generates only one attribute for a combination of get/set. For example, if an attribute name is ID, the methods would be getID() and setID(). The business object would have one attribute with name ID and ASI as getter=getID();setter=setID(), type=int, name=ID.

Table 7 describes the ASI for non-method attributes.

Table 7. Attribute-level ASI for non-method attributes

Attribute	Description
Name	Specifies the business object field name
Type	Specifies the business object field type
MaxLength	255 characters, by default
IsKey	Set to false.
IsForeignKey	Set to false.
IsRequired	Set to false. Set to true if the field is mandatory.
AppSpecificInfo	This attribute is formatted as follows: name=; type=; use_attribute_value=busobj.attrname(optional); getter=; setter=;
DefaultValue	None

Table 8 describes the ASI for non-method attributes.

Table 8. Attribute-level ASI for method attributes

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

Table 8. Attribute-level ASI for method attributes (continued)

Attribute	Description
Relationship	If the child is a container attribute, this is set to Container.
IsKey	Set to false.
IsForeignKey	Set to false.
IsRequired	Set to false.
AppSpecificInfo	None
Cardinality	1

Application-specific information for XML list business object functions

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

Application-specific information can be specified for the overall business object as well as for each business object attribute.

Business object-level ASI

The business object ASI is type=XMLList. If the table type is not TABLE_CONVERSION, the following ASI must be present:

- TN=<table name>: The name of the table from which data has to be fetched.
- TABLE_TYPE=<table type>: The type of the table.

  The table type can be one of the following:
  - OWTABLE
  - OWVIEW
  - FOREIGN_TABLE

Verb ASI

This property is not used for XML business objects.

Attribute-level ASI

The attributes of the business objects map to specific columns of a table.

The following ASI are set for the following attributes:

- alias=<column name>: The alias of the column that is being fetched.
- Name=<column name>: The name of the column in the component.
- type=<type>: The data type, this property would be used in preparing the where clause.
- operator_type=<Type of operand>. For example, GT, LT. This value is used in comparing the values when the where clause is executed.
- table=<table name>: The name of the table to which the column belongs. This ASI is used only when data is fetched from a table conversion process.
- clause_type: The kind of clause, either WHERE, AND or OR.
- clause_seq: A number representing the order in which the columns are added in the where clause

- sorting=<ASCD, DESD>: If the value is ASCD the data sequencing uses ascending for this column, if the value is DESD it uses descending for this column.

Business object handler

The generic OneWorld business object handler handles processes that might call a set of business functions on components that are exposed through OneWorld components. It can also fetch data from OneWorld tables using the XML List feature of OneWorld. When a new business object enters the business object handler, it reads the business object ASI to determine if the adapter should invoke a business function or if it should create an XML document for XML List. The ASI tag `type` specifies the type of call. If `type=BFN`, then the adapter passes the call to the business function invoker that instantiates OneWorld Java objects and invokes business functions. If the ASI has `type=XMLList`, then the adapter generates the XML document for the `CreateList` API.

Sample business object

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

The samples provided in this section are as follows:

- "Sample GenJava script file"
- "Business object structure for the above example" on page 45

Sample GenJava script file

OneWorld provides a utility, GenJava, that generates Java wrappers for the business functions running as part of the OneWorld server. GenJava requires a script file, written using iJDEScript. The following example uses the script file, `AddressBook.cmd`. `AddressBook.cmd` specifies the library and the interface to which the set of business functions is modularized.

Once GenJava is executed, it creates Java class files for all the interface classes and associated data structures. GenJava compiles the generated Java files, generates Java docs, and packages them into two .jar files, one for Java classes and one for Java doc. The below sample renders `AddressBookInterop.jar` and `AddressBookInteropDoc.jar` files.

To execute the following sample, type the following from the command line:

```
GenJava /UserID JDE /Password JDE /Environment JDETest /cmd AddressBook.cmd
```

There are options available for running GenJava. GenJava is present in the `<INSTALL>\system\bin32 folder`.

Please refer to the section on Running GenJava in the Interoperability Guide, provided by OneWorld. The sample GenJava script file is shown below:

```
# This example creates a library whose name is derived from an input parameter
# (library) if one is specified. A default value is used otherwise.

define library JDEAddressBook

login
```

```
library JDEAddressBook
   interface AddressBook
      import B0100031
      import B0100019
      import B0100032
      import B0100002
      import B0100033
build
logout
```

While preparing this script, consider the mapping of the Interface class to a business object in WebSphere Business Integration Adapter and relate the business functions as method sequences that are required for performing the intended actions for a verb. For example, if a business object is a `SalesOrder` business object, then the interface `SalesOrder` in the script file must include all the business functions that it needs to perform actions on the `SalesOrder` object through WebSphere Business Integration Adapter. The sequence of method execution for each verb is accomplished by populating the verb ASI for the business object. You must be able to do this in the business object generation process using the ODA. You can also edit the verb ASI using Business Object Designer after the business objects have been generated.

Business object structure for the above example

The following figure shows the business object structure for the above example in Business Object Designer.

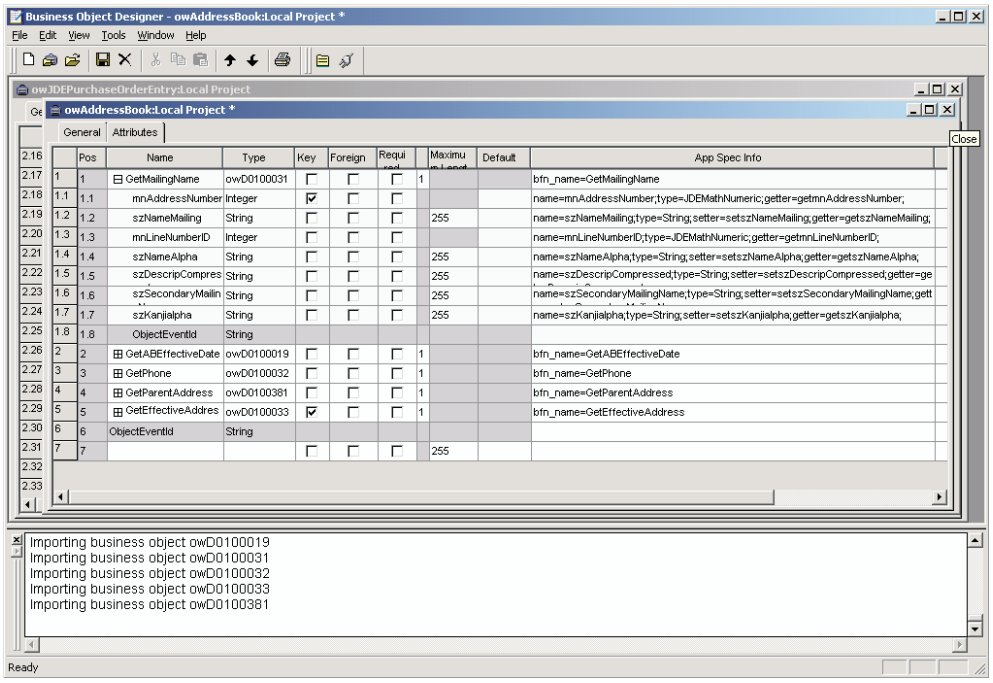

Figure 15. Business object structure for the AddressBook example

This section explains the business object structure for the `AddressBook` example, in the following sections:

AddressBook

```
Name AddressBook
ASI (type=BFN; class_name=com.JD Edwards.interop.AddressBook)
```

Attributes

The AddressBook business object includes the following objects:

- GetMailingName (Object)
- GetABEffectiveDate (Object)
- GetPhone (Object)
- GetParentAddress (Object)
- GetEffectiveAddress (Object)

Verb ASI

The Verb ASI uses the following Retrieve and RetrieveDetails objects:

- Retrieve — GetEffectiveAddress
- RetrieveDetails — GetPhone; GetMailingName

D0100033

```
Name D0100033
ASI (type=BFN;
class_name=com.JD Edwards.interop.jdeaddressbook.D0100033;
bfn_name=GetEffectiveAddress)
```

Attributes

The D0100033 business object includes the following objects:

- mnAddressNumber (Integer)
  (ASI) type = JDEMathNumeric; name = m_mnAddressNumber;
  use_attribute_value=;getter=getmnAddressNumber;setter=;
- jdDateBeginningEffective (Date)
  (ASI) type = JDEDate; name = m_mnAddressNumber;
  use_attribute_value=;getter=getjdDateBeginningEffective;
  setter=setjdDateBeginningEffective;

Verb ASI

None.

Generating business objects

Each time an event occurs during run time, a OneWorld application sends a message object containing object-level data and information about the type of transaction. The connector maps this data to the corresponding business object definition, to create an application-specific business object. The connector sends these business objects on to the integration broker for processing. It also receives business objects back from the integration broker, which it passes back to the OneWorld application.

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

Business Object Designer provides a graphical interface that enables you to create and modify business object definitions for use at run time. For details, see Chapter 4, “Creating and modifying business objects,” on page 23.

Chapter 6. Error handling and event codes

This chapter describes how the adapter for JD Edwards OneWorld handles errors and event codes. The adapter generates logging and tracing messages. This chapter describes these messages. The chapter contains the following sections:

Error handling

This section describes error handling for the OneWorld adapter and the OneWorld ODA.

Adapter

The adapter throws the following three types of exceptions when it executes a business function in OneWorld:

- Fatal
- Recoverable
- Reject

Fatal

`FatalException` class conditions require manual intervention. The adapter catches fatal exception conditions and writes the text of the exception in the `ReturnStatusDescriptor` string. The returned status is `FAIL`.

Recoverable

In the case of a recoverable error, the adapter tries to perform the execution of the business function again. If the Recoverable exception is thrown again, the adapter writes the text of the exception in the `ReturnStatusDescriptor` string. The returned status is `FAIL`. If the second try is successful, the returned status is `VALCHANGED`.

Reject

In the case of a Reject exception, the return value determines if it is an error or a warning. The possible values are as follows:

- Successful=0—This status would not be returned if an exception is raised.
- Warning=1—The adapter populates the `ReturnStatusDescriptor` with the exception message and returns a status of `VALCHANGED`.
- Error=2—The adapter populates the `ReturnStatusDescriptor` with the exception message and the returns a status of `FAIL`.

For XMLList business object processing, the response XML document contains the error codes and error strings in case of errors. The adapter writes the error codes and error strings in the `ReturnStatusDescriptor` and returns a status of `FAIL`.

ODA

The OneWorld ODA throws an exception in the following scenarios:

- If the path specified for the .jar file does not exist or cannot be accessed

- If the .jar file is corrupt or cannot be accessed
- If the .jar file is empty

ODK properties define the trace file name and trace level. The ODK wizard manages these two properties. The trace file can be found in the OneWorld folder of the Crossworlds/ODA folder. The default name of the file is `OneWorldODAtrace.txt`. The message file containing the error and trace messages has the following naming convention:

```
BIA_<ODAAgentName>Agent.txt
```

`ODAAgentName` is the value from the variable of the same name found in the `start` file for the ODA. If you change the value of the `ODAAgentName` variable, then you must also change the message file name. The error and trace message file is in the ODA messages folder.

See *Tracing Exceptions and Messages* in the *Business Object Development Guide* for more information on the trace file and the message file.

Corrupt records in database

If the various software layers in EnterpriseOne are not properly synchronized after you install new custom JD Edwards business functions (an IBM WebSphere EnterpriseOne Adapter prerequisite), improper data field mapping can occur when manipulating records in the database. This usually happens when a table is defined using all the correct steps, but then later altered to reflect a change in the table. If a mistake is made in the original adapter installation steps, but not detected until after the supporting adapter tables are generated, the JD Edwards administrator must resynchronize the EnterpriseOne software layers.

Improper mapping usually presents the following symptoms:

- Records that contain some fields with proper information and others with bad data
- Records with fields that appear to be swapped
- Records with fields that are missing or truncated.

To resynchronize the EnterpriseOne software layers (for version 8.9.5 and application version 8.11), perform the following steps.

Note: It is important to follow the step sequence as shown. Refer to JD Edwards documentation for different versions of JD Edwards products.

1. Completely regenerate the database table using the EnterpriseOne OMW tooling

   Note: This delete and add operation removes all existing records in the table.
2. Regenerate the header file that defines the "typedef" data structure to contain records from the table. There is a distinct function to perform this task. (Header file regeneration is implicitly performed when the table is first defined and saved, but not when the table is simply modified.)
3. Recompile all business functions that reference this table. This task provides the business functions with the latest header file definitions.
4. Completely close all EnterpriseOne products (to clear any old cached data structures).

5. Delete all EnterpriseOne 8.95 TAM spec files (glbltbl.xdb, glbltbl.ddb, dddict.xdb, dddict.ddb, ddtext.xdb, ddtext.ddb).
6. Re-deploy and re-test the new business functions.

Undefined class errors

The jar files that support EnterpriseOne interoperability occasionally change in new releases of EnterpriseOne. Unfortunately the list of required jar files is incomplete or inaccurate in some older versions of EnterpriseOne (8.94 and 8.95). For example the list of jar files required for 8.9.5 is as follows:

- ApplicationAPIs_JAR.jar
- ApplicationLogic_JAR.jar
- Base_JAR.jar
- BizLogicContainerClient_JAR.jar
- BizLogicContainer_JAR.jar
- castor.jar
- Connector.jar
- EventProcessor_EJB.jar
- EventProcessor_JAR.jar
- JdbjBase_JAR.jar
- JdbjInterfaces_JAR.jar
- JdeNet_JAR.jar
- log4j.jar
- PMApi_JAR.jar
- Spec_JAR.jar
- System_JAR.jar
- xerces.jar

This list was obtained through JD Edwards support but did not appear in JD Edwards version 8.9.5 documentation. Contact IBM support for further information or help.

Logging

All messages are read from the message files `BIA_<ODAAgentName>Agent.txt` for the ODA and `BIA_OneWorldAdapter.txt` for the adapter.

Tracing

Tracing is an optional debugging feature you can turn on to closely follow connector behavior. Trace messages, by default, are written to STDOUT. For more on configuring trace messages, see the connector configuration properties in "Using log and trace files" on page 21. For more information on tracing, including how to enable and set it, see the *Connector Development Guide*.

Table 9 on page 52 lists the recommended content for connector tracing message levels.

Table 9. Tracing messages content

Level	Description
Level 0	This level is used for trace messages that identify the connector version. No other tracing is performed at this level.
Level 1	Use this level for trace messages that: • Provide status information. • Provide key information on each business object processed. • Record each time a polling thread detects a new message in an input queue.
Level 2	Use this level for trace messages that: • Identify the business object handler used for each object that the connector processes. • Log each time a business object is posted to the integration broker. • Indicate each time a request business object is received.
Level 3	Use this level for trace messages that: • Identify the sub-objects being processed, if applicable. These messages appear when the connector has encountered a foreign key in a business object or when the connector sets a foreign key in a business object. • Relate to business object processing. Examples of this include finding a match between business objects, or finding a business object in an array of child business objects.
Level 4	Use this level for trace messages that: • Identify application-specific information. Examples of this include the values returned by the methods that process the application-specific information fields in business objects. • Identify when the connector enters or exits a function. These messages help trace the process flow of the connector. • Record any thread-specific processing. For example, if the connector spawns multiple threads, a message logs the creation of each new thread.
Level 5	Use this level for trace messages that: • Indicate connector initialization. This type of message can include, for example, the value of each connector configurator property that has been retrieved from the broker. • Detail the status of each thread that the connector spawns while it is running. • Represent statements executed in the application. The connector log file contains all statements executed in the target application and the value of any variables that are substituted, where applicable. • Record business object dumps. The connector should output a text representation of a business object before it begins processing (showing the object that the connector receives from the collaboration) as well as after it finishes processing the object (showing the object that the connector returns to the collaboration).

Event status codes

The adapter issues event status codes that are described in the following list.

- `READY_FOR_POLL=0` The event is ready to be picked up by the next poll call. The events are created with this status in the OneWorld application. The connector looks for events with READY_FOR_POLL status while fetching the events from the application during pollForEvents processing.

- IN_PROGRESS=1 The event has been picked up and is sent to the connector framework. The connector changes the status of the event to IN_PROGRESS after it picks the event for processing.
- UNSUBSCRIBED=2 The event has not been subscribed to. The connector sets the status to UNSUBSCRIBED if the isSubscribed call returns a false value.
- SUCCESS=3 The event was successfully processed by the connector framework. The connector sets the status to SUCCESS if the event is processed successfully by the connector framework.
- ERROR_PROCESSING_EVENT=-1 There was an error processing the event.
- ERROR_POSTING_EVENT=-2 There was an error posting the event to the connector framework. This status is set if the call to gotApplEvent fails during pollForEvents processing.
- ERROR_OBJECT_NOT_FOUND=-3 The object for which the event was created could not be found. This status is set if the doVerbFor call could not find the object during pollForEvents processing.
- INTERNAL ADAPTER CODE=5 The status of 5 indicates a configuration problem. One possible reason is an incorrect data structure defined for the event table. A possible solution is to correct the data structure by manually modifying the business functions for the event table (B551005,B551006,B551007, B551008, B551009). The modification is needed only for the .h file. Once the modification is made, the package must be rebuilt and deployed. For more information about how to do this, see "Installing and configuring IBM event store" on page 19.

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 10 on page 57.)

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 10 on page 57.

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 10 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 10, 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 10. 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 10. 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 10. 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 10. 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 10. 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 10. 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 10. 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 66.

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 80).

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 81 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 86.)

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 81.
- 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 89..
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 88.

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 56.

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>
```

Startup scripts for Adapter Framework 2.6

This appendix describes changes that occurred with the release of Adapter Framework 2.6 and includes sample startup scripts for the adapter and the ODA. The topics covered in this appendix are the following:

Overview of Adapter Framework 2.6 changes

If you are running the Adapter for JD Edwards OneWorld with WebSphere Business Integration Adapter Framework 2.6, you must modify startup scripts after installing, but before starting, the connector. Otherwise, the Adapter for JDE OneWorld may not work with the new framework.

The changes to startup scripts for both the adapter and the ODA are required because .jar files and environment variables were renamed in Adapter Framework 2.6. Refer to the document *Installing WebSphere Business Integration Adapters, Version 2.6* for details on how to install Adapter Framework 2.6. Then install the Adapter for JDE OneWorld as documented in Chapter 2, "Installing the adapter," on page 11. Before replacing startup scripts and environment variables, see the document *Migrating Adapters to Adapter Framework, Version 2.6*, section "Adapter Start Scripts, Object Discovery Agent Start Scripts" for an overview of changes in the start scripts.

Samples of the modified startup scripts are shown in subsequent sections. To obtain copies of these files, see the Technote *Adapter for JDE OneWorld startup scripts must be modified to run with WebSphere Business Integration Adapter Framework 2.6*

start_OneWorld.bat for Adapter Framework 2.6

Here is a sample of the startup script modification for Windows:

```
REM @echo off
setlocal
REM Start - Adapter Specific Variables
REM set the name to be the application connector that is starting
 set CONNAME=%1
 set CONNPACKAGENAME=com.ibm.adapters.oneworld.OneWorldAdapterAgent
REM set the server name to be the interchange that is being targeted
 set SERVER=%2
REM End - Adapter Specific Variables
REM Branch between WBIA_RUNTIME and CROSSWORLDS
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\CwConnEnv.bat
REM set the directory where the specific connector resides
 set CONNDIR="%WBIA_RUNTIME%"\connectors\%1
REM set the AGENT - the name of the adapter jar
```

```
 set AGENT=%CONNDIR%\BIA_%CONNAME%.jar
REM Go to the adapter specific drive and directory
 cd /d %CONNDIR%
REM Set JVMArgs variable
 set JVMArgs=-Dconfig_file="%CONNDIR%"\dependencies\jdeinterop.ini
REM Set JCLASSES - to be set in CLASSPATH
 set JCLASSES=.;%AGENT%;%JCLASSES%
Rem Set ExtDirs - to have the dependencies folder
        set ExtDirs=%CONNDIR%\dependencies;%CONNDIR%\repository;%ExtDirs%
call "%WBIA_RUNTIME%\bin\start_adapter" -n%CONNAME% -s%SERVER%
-l%CONNPACKAGENAME% %3 %4 %5
goto END
:CROSSWORLDS
REM All the old functionality goes here
REM call CWConnEnv
 call "%CROSSWORLDS%"\bin\CWConnEnv
REM set the directory where the specific adapter resides
 set CONNDIR="%CROSSWORLDS%"\connectors\%1
REM set the AGENT - the name of the adapter jar
 set AGENT=%CONNDIR%\BIA_%CONNAME%.jar
REM Go to the adapter specific drive & directory
 cd /d %CONNDIR%
REM Set JCLASSES - to be set in CLASSPATH
 set JCLASSES=.;%JCLASSES%;%AGENT%;
REM config file location defaults to HOME\InterchangeSystem.cfg on the local machine
REM start the connector dll under the Java Application End
%CWJAVA% -mx128m -Dconfig_file="%CONNDIR%"\dependencies\jdeinterop.ini
-Djava.ext.dirs="%MQ_LIB%";%JRE_EXT_DIRS%;%CONNDIR%\
repository;%CONNDIR%\dependencies -Djava.library.path="%CROSSWORLDS%"\
bin;%CONNDIR%;"%MQ_LIB%";%JRE_EXT_DIRS%;%CONNDIR%\repository;%CONNDIR%
\dependencies %ORB_PROPERTY% -Duser.home="%CROSSWORLDS%" -cp %JCLASSES%;
 AppEndWrapper -l%CONNPACKAGENAME% -n%CONNAME%Connector -s%SERVER%  %3 %4 %5
endlocal
pause
```

start_OneWorld.sh for Adapter Framework 2.6

Here is a sample of the startup script modification for UNIX:

```
#!/bin/sh -x # This script should be called from connector_manager.sh, and
accepts the following parameters: #  start_myconnector.sh <CONNECTORNAME>
<SERVERNAME> <OPTIONAL ADDITIONAL PARAMETERS> # #  Any line that has the
#   <-- Check on it should be reviewed for your connector #
############################################## # General connector
variable initialization section # No changes should be necessary # set
CONNECTOR_TYPE  -- This is needed to differentate Java from C++ conncetors
# set CONDIR  the directory where the specific connector resides, then
change to this directory # set CONNAME the name to be the application
connector that is starting # set SERVER  the server name to be the
interchange that is being targeted # set CONJAR to the fullpath of the
connector .jar file. CONNAME=$1 SERVER=$2 . ${CROSSWORLDS}/bin/CWConnEnv.sh
CONDIR=${CROSSWORLDS}/connectors/${CONNAME} CONJAR=${CONDIR}/
BIA_${CONNAME}.jar ############################################## #
Connector specific variable initialization section # Set your Connector
agent here #Comment one of the next two sections in: #For C++ connectors
comment in the next two lines #CONNECTOR_TYPE="-d"
#CONNECTOR_AGENT=${CONNAME} #For Java connectors comment in the next two
lines    CONNECTOR_TYPE="-l"
CONNECTOR_AGENT=com.ibm.adapters.oneworld.OneWorldAdapterAgent # Set any
connector specific Variables here, add the module jar files generated using
GenJava #CON_SPEC_JAR_ONE=${CONDIR}/dependencies/Connector.jar:${CONDIR}/
```

```
dependencies/Kernel.jar:${CROSSWORLDS}/lib/xerces.jar:${CONDIR}/
dependencies/IBMEventsInterop.jar: # Set Conn specific classpath here #Any
specific variables defined above need to be added to the
CON_SPECIFIC_CLASSPATH below.
CON_SPECIFIC_CLASSPATH=${CON_SPEC_JAR_ONE}                    #   <-- Check
# Set any Connector specific start options here # CON_START_OPTIONS="
-tMAIN_SINGLE_THREADED " # We are using the JVM_FLAGS that are aet in the
CWSharedenv.sh file, if you need to use different flags, #  Change
$JVM_FLAGS to be what you need to use. # No changes should be necessary
below this line. if [ -f ${CROSSWORLDS}/wbiart/wbiart.jar ] then
JCLASSES=${CONJAR}:${CON_SPECIFIC_CLASSPATH}:${CWCLASSES}:${CONDIR}:${JCLASSES}
export JCLASSES JVMArgs=-Dconfig_file=${CONDIR}/dependencies/jdeinterop.ini
export JVMArgs ExtDirs=${CONDIR}/dependencies:${CONDIR}/
repository:${ExtDirs} export ExtDirs exec start_adapter.sh
${CONNECTOR_TYPE}${CONNECTOR_AGENT} -n${CONNAME}Connector -s${SERVER}
${CON_START_OPTIONS} $3 $4 $5 $6 $7 $8 $9 else CWCLASSES=${CWCLASSES}
CLASSPATH=${CONJAR}:${CON_SPECIFIC_CLASSPATH}:${CWCLASSES}:${CONDIR} echo
$CLASSPATH ${CROSSWORLDS}/bin/check_path.sh "$CLASSPATH" exec ${CWJAVA}
${JVM_FLAGS} -Dconfig_file=${CONDIR}/dependencies/jdeinterop.ini
-Djava.library.path=${LD_LIBRARY_PATH}:${CONDIR}/dependencies:${CONDIR}/
repository -Djava.ext.dirs=${JRE_EXT_DIRS}:${CONDIR}/
dependencies:${CONDIR}/repository -classpath ${CLASSPATH} AppEndWrapper
${CONNECTOR_TYPE}${CONNECTOR_AGENT} -n${CONNAME}Connector -s${SERVER} $3 $4
$5 $6 $7 $8 $9 fi
```

start_OneWorldODA.bat for Adapter Framework 2.6

Here is a sample of the startup script modification for the object discovery agent runing on Windows:

```
REM set AGENTNAME=OneWorldODA
REM set AGENT="%CROSSWORLDS%"\ODA\OneWorld\BIA_OneWorldODA.jar
REM set AGENTCLASS=com.ibm.oda.oneworldoda.OneWorldOda
setlocal
REM IF WBIA_RUNTIME is set use ODK Runtime scripts
if "%WBIA_RUNTIME%"=="" goto CROSSWORLDS
REM Invoke the CWODAENV.bat
call "%WBIA_RUNTIME%"\bin\CWODAEnv
goto LAUNCH
:CROSSWORLDS
REM Invoke the CWODAENV.bat
call "%CROSSWORLDS%"\bin\CWODAEnv
:LAUNCH
REM Define local batch PATH to insure we execute our jre
set PATH="%CROSSWORLDS%";"%CROSSWORLDS%"\bin;%PATH%
set AGENTNAME=OneWorldODA
set AGENT="%CROSSWORLDS%"\ODA\OneWorld\BIA_OneWorldODA.jar
set AGENTCLASS=com.ibm.oda.oneworldoda.OneWorldOda
set CONDIR="%CROSSWORLDS%"\connectors\oneworld
set JCLASSES=%JCLASSES%;%AGENT%
REM Start the Object Discovery Agent
%CWJAVA% -Duser.home="%CROSSWORLDS%" -
Djava.library.path=%JLIBRARIES%;%CONDIR%\dependencies;%CONDIR%\repository -
Djava.ext.dirs="%JRE_EXT_DIRS%";%CONDIR%\dependencies;%CONDIR%\repository -mx128m
%ORB_DEP% -classpath %JCLASSES% com.crossworlds.ODKInfrastructure.%CONNECTION% -
l%AGENTNAME% -c%AGENTCLASS%
endlocal
pause
```

oda.dd.xml for Adapter Framework 2.6

In addition to modifying the startup script for the ODA, you must add the following file (oda.dd.xml) under %WBIA_RUNTIME%\ODA\OneWorld for the ODA to work:

```
<?xml version="1.0" encoding="utf-8" ?> - <oda> - <startup>
<messagefile prefix="BIA_" />    </startup>   </oda>
```

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.

Each copy or any portion of these sample programs or any derivative work, must include a copyright notice as follows: &copr; (your company name) (year). Portions of this code are derived from IBM Corp. Sample Programs. &corp; Copyright IBM Corp. _enter the year or years_. All rights reserved.

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

IBM and related trademarks:
http://www.ibm.com/legal/copytrade.shtml

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/).

Index

A

Printed in USA