

IBM WebSphere Business Integration Adapters



Adapter for JDBC User Guide

V 24.x

IBM WebSphere Business Integration Adapters



Adapter for JDBC User Guide

V 24.x

Note!

Before using this information and the product it supports, read the information in "Notices" on page 121.

19December2003

This edition of this document applies to the adapter for JDBC version 2.4.x and to all subsequent releases and modifications until otherwise indicated in new editions.

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

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

© Copyright International Business Machines Corporation 2000, 2003. All rights reserved.

US Government Users Restricted Rights – Use, duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp.

Contents

Figures	v
About this document	vii
Audience	vii
Related documents	vii
Typographic conventions	vii
New in this release	ix
New in release 2.4.x	ix
New in release 2.3.x	ix
New in release 2.2.x	x
New in release 2.1.x	x
New in release 2.0.x	x
New in release 1.9.x	x
New in release 1.8.x	xi
New in release 1.7.x	xi
New in release 1.6.x	xi
New in release 1.5.x	xii
Chapter 1. Overview of the connector	1
Connector components	1
How the connector works	2
Chapter 2. Installing and configuring the connector	7
Adapter environment	7
Prerequisites	8
Installing the adapter and related files	9
Installed file structure	9
Enabling the application for the connector	10
Enabling multi-driver support	13
Enabling the custom business object handler class	14
Configuring the connector	14
Creating multiple instances of the connector	26
Starting the connector	28
Stopping the connector	29
Chapter 3. Understanding business objects for the connector	31
Business object and attribute naming conventions	31
Business object structure	31
Business object verb processing	36
Business object attribute properties	50
Business object application-specific information	53
Chapter 4. Generating business object definitions using JDBCODA	65
Installation and usage	65
Using JDBCODA in business object designer	68
Contents of the generated definition	74
Sample business object definition file	77
Inserting attributes that contain child business objects	78
Adding information to the business object definition	78
Chapter 5. Troubleshooting and error handling	79
Startup problems	79
Event processing	79

Mapping (ICS Integration Broker only)	79
Error handling and logging	80
Loss of connection to the application	82
Fetch out-of-sequence error	82
Inability to locate event or archive tables when DB2 is used	82
Enabling the connector to work with a DB2 database	82
Resource-busy error	83
JDBCODA behaves improperly because of unsupported JDBC driver	83
Appendix A. Standard configuration properties for connectors	85
New and deleted properties	85
Configuring standard connector properties	85
Summary of standard properties	86
Standard configuration properties	90
Appendix B. Connector Configurator	101
Overview of Connector Configurator	101
Starting Connector Configurator	102
Running Configurator from System Manager	103
Creating a connector-specific property template	103
Creating a new configuration file	105
Using an existing file	106
Completing a configuration file	107
Setting the configuration file properties	108
Saving your configuration file	113
Changing a configuration file	114
Completing the configuration	114
Using Connector Configurator in a globalized environment	114
Appendix C. Business object samples.	117
AfterUpdateSPSampleBO.txt	117
BeforeCreateSPSampleBO.txt	117
BOwithDifferentParameterOrder.txt	117
BOwithIOandOPParams.txt	118
BOwithFewerSPParamsthanBOAttribs.txt	118
CreateSPUpdateSPSampleBO.txt	118
Appendix D. Support for null and blank values	119
Pass and Fail Scenarios	119
Functionality	120
Notices	121
Programming interface information	122
Trademarks and service marks	122

Figures

1. Business object request architecture	2	6. Selecting the ODA	69
2. Typical single-cardinality relationship	33	7. Configuring agent initialization properties	70
3. Multiple-cardinality business object relationship	35	8. Tree of Schema with Expanded Nodes	71
4. Single-cardinality business object with relationship stored in the child	35	9. Confirming Selection of Database Objects	72
5. Example of relationships among business objects	58	10. Associating Stored Procedures with Stored Procedure Attributes	73

About this document

The IBM[®] WebSphere[®] Business Integration Adapter portfolio supplies integration connectivity for leading e-business technologies, enterprise applications, and 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, and business object development for the adapter for JDBC.

Audience

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

Related documents

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

You can install related documentation from the following sites:

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

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

Typographic conventions

This document uses the following conventions:

<code>courier font</code>	Indicates a literal value, such as a command name, file name, information that you type, or information that the system prints on the screen.
bold	Indicates a new term the first time that it appears.
<i>italic</i>	Indicates a variable name or a cross-reference.

<i>blue outline</i>	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.
<i>ProductDir</i>	Represents the directory where the IBM WebSphere Business Integration Adapters product is installed. The default product directory is WebSphereAdapters.
{ }	In a syntax line, curly braces surround a set of options from which you must choose one and only one.
	In a syntax line, a pipe separates a set of options from which you must choose one and only one.
[]	In a syntax line, square brackets surround an optional parameter.
...	In a syntax line, ellipses indicate a repetition of the previous parameter. For example, <code>option[...]</code> means that you can enter multiple, comma-separated options.
< >	Angle brackets surround individual elements of a name to distinguish them from each other, as in <code><server_name><connector_name>tmp.log</code> .
/, \	In this document, backslashes (\) are used as the convention for directory paths. For UNIX installations, substitute slashes (/) for backslashes. All product pathnames are relative to the directory where the connector for JDBC is installed on your system.
UNIX:/Windows:	Paragraphs beginning with either of these indicate notes listing operating system differences.
<i>%text%</i> and <i>\$text</i>	Text within percent (%) signs indicates the value of the Windows text system variable or user variable. The equivalent notation in a UNIX environment is <i>\$text</i> , indicating the value of the <i>text</i> UNIX environment variable.

New in this release

New in release 2.4.x

Updated in December 2003.

Beginning with version 2.1.0, the adapter for JDBC is no longer supported on Microsoft Windows NT.

The release of this document for adapter version 2.4.x also contains the following new or corrected information:

- Information formerly in Chapter 2 about installing the connector has been removed. See that chapter for the new location of that information.
- Under “Configuring the connector” in Chapter 2, the connector-specific configuration properties of ApplicationPassword and ApplicationUserName are not required when trusted authentication is being used.
- Under “Business object verb processing” in Chapter 3, the instructions for DeltaUpdate operations have been added and the instructions for Delete operations have been revised.
- Under “Business object application-specific information” in Chapter 3, information has been added to describe how a CLOB datatype would be defined.
- In the same section of Chapter 3 under “Generating a business object’s unique identifier,” information regarding IBM DB2 has been added.
- Stored procedures have been added to the section in Chapter 4, “Expanding nodes and selecting tables, views and stored procedures.” And, details about stored procedure attributes have been added under “Providing additional information.”

New in release 2.3.x

Updated in July 2003. The release of this document for adapter version 2.3.x contains the following new or corrected information:

- The adapter can now use WebSphere Application Server as an integration broker.
- The adapter now runs on the following platforms:
 - HP-UX11i
 - AIX 5.x
 - Solaris 7 and 8
- Support for return of result set from Oracle stored procedures has been added.
- Support for CLOB data types has been added.
- Grandparent access support for copy attributes has been added. Copy attributes can now be accessed from the parent, which allows propagation of attributes down the business object hierarchy.
- The restriction has been removed that an eventId must be a numeric data type.

New in release 2.2.x

Updated in March 2003. The “CrossWorlds” name is no longer used to describe an entire system or to modify the names of components or tools, which are otherwise mostly the same as before. For example “CrossWorlds System Manager” is now “System Manager,” and “CrossWorlds InterChange Server” is now “WebSphere InterChange Server.”

The release of this document for connector version 2.2.x contains the following new or corrected information:

- Support has been added for the following:
 - Wrapper objects at the top-level of business objects
 - LIKE operator
 - Hex/binary data
 - Stored procedures for the RetrieveUpdate verb
 - Verb Application Specific Information for RetrieveByContent
 - Verb Application Specific Information in the WHERE clause when the WHERE clause length is 0 in RetrieveByContent
- The ConnectorID property has been changed from an int to a String to allow for a more descriptive name.
- The DRIVERLIB variable has been added to point to the native libraries used by the custom JDBC drivers.
- More functionality has been added to check for loss of database connectivity during object processing.
- The Schema Name property is now used when retrieving and archiving events.
- Sample business objects have been added, RetrieveResultSet_SampleBOwithMcardChild and RetrieveResultSet_SampleBOwithScardChild, (located in the \connectors\JDBC\Samples directory).

New in release 2.1.x

The orjdbcobjconverter.pl Perl script (used to convert an Oracle object to a JDBC-specific object) is no longer installed with the connector. Instead, you can obtain this script from the IBM eCare support Web site (<http://www.ibm.com/software/integration/cw/support>).

New in release 2.0.x

The connector has been internationalized. For more information, see “Processing Locale-Dependent Data,” on page 6 and Appendix A, Standard Configuration Properties for IBM CrossWorlds Connectors

New in release 1.9.x

The IBM WebSphere Business Integration Adapter for JDBC includes the connector for JDBC. This adapter supports two integration brokers: InterChange Server (ICS) and WebSphere MQIntegrator. An integration broker is an application that performs integration of heterogeneous sets of applications; it provides services such as data routing.

The IBM WebSphere Business Integration Adapter for JDBC includes the following:

- An application component specific to JDBC
- JDBCODA
- A sample business object (located in the \connectors\JDBC\Samples directory)
- IBM WebSphere Adapter Framework, which consists of the following:
 - Connector Framework
 - Development tools (including Business Object Designer and Connector Configurator)
 - APIs (including ODK, JCDK, and CDK)

This manual provides information about using the adapter with both the ICS and WebSphere MQIntegrator integration brokers.

Note: Because the connector has not been internationalized, do not run it against InterChange Server version 4.1.1 if you cannot guarantee that only ISO Latin-1 data will be processed.

New in release 1.8.x

JDBCODA, which generates business object definitions for the connector, has been enhanced, and the documentation of it has been improved. See Chapter 4, “Generating business object definitions using JDBCODA,” on page 65.

New in release 1.7.x

CrossWorlds installation now provides the IBM branded JDBC driver for the MS SQL Server to replace the WebLogic JDBC driver, and continues to provide the Oracle thin driver.

New in release 1.6.x

The release of this document for connector version 1.6.x contains the following new or corrected information:

- An Object Discovery Agent utility has been developed to create business objects for the connector. This should be used instead of the JDBCBOGEN provided with earlier connector releases. See Chapter 4, “Generating business object definitions using JDBCODA,” on page 65.
- The CheckForEventTableInInit property description has been added to the “Connector-specific properties” on page 15.
- The CloseDBConnection property description has been added to the “Connector-specific properties” on page 15.
- The SPBeforePollCall property description has been added to the “Connector-specific properties” on page 15.
- Support has been added for stored procedures returning ResultSet. See “Business object retrieve operations” on page 48 and “Business object RetrieveByContent operations” on page 49.
- Support has been added for fixed length string attributes. See the description in Table 11 on page 55.
- Support has been added for a custom BO handler.

New in release 1.5.x

The release of this document for connector version 1.5.x contains the following new or corrected information:

- The `ArchiveTableName` property description has been updated. See the “`ArchiveTableName`” on page 17 for details.
- The `AutoCommit` property has been added to the “Connector-specific properties” on page 15.
- The `DateFormat` property has been added to the “Connector-specific properties” on page 15.
- The `EventKeyDel` property has been updated to accommodate name value pairs. See “`EventKeyDel`” on page 20 for details.
- The `EventQueryType` property has been added to the “Connector-specific properties” on page 15.
- The `PingQuery` property has been added to the “Connector-specific properties” on page 15.
- The `PreserveUIDSeq` property has been added to the “Connector-specific properties” on page 15.
- The `SchemaName` property has been added to the “Connector-specific properties” on page 15.
- Support for the stored procedures for the `RetrieveByContent` verb. See “Business object `RetrieveByContent` operations” on page 49 for details.
- Support for retrieving business objects, based on non-key values, for Event Processing. See “Retrieving business objects for event processing” on page 4 for details.

Chapter 1. Overview of the connector

This chapter describes the connector component of the IBM WebSphere Business Integration Adapter for JDBC.

This chapter contains the following sections:

- “Connector components”
- “How the connector works” on page 2

Connector components

Connectors consist of two parts: 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 application or technology (in this case, JDBC). 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

The connector for JDBC enables the integration broker to exchange business objects with an application built on any database supported by a driver that follows the JDBC 2.0 or above specification. This section presents a high-level description of the connector’s architecture and use of different JDBC drivers.

For specifying the driver to be used by the connector to connect to the database, see “Enabling multi-driver support” on page 13.

The connector connects to the application database by using the JDBC Connect mechanism. One connector-specific configuration parameter (`DatabaseURL`) allows you to specify the name of the database server to which the connector should connect. For information on the configuration parameters, see “Configuring the connector” on page 14.

When the connector is started, it establishes a connection pool with the database. It uses connections from this pool for all transaction processing with the database. On termination of the connector, all connections in the pool are closed.

Connector architecture

Figure 1 shows the connector components and their relationships within the business integration system.

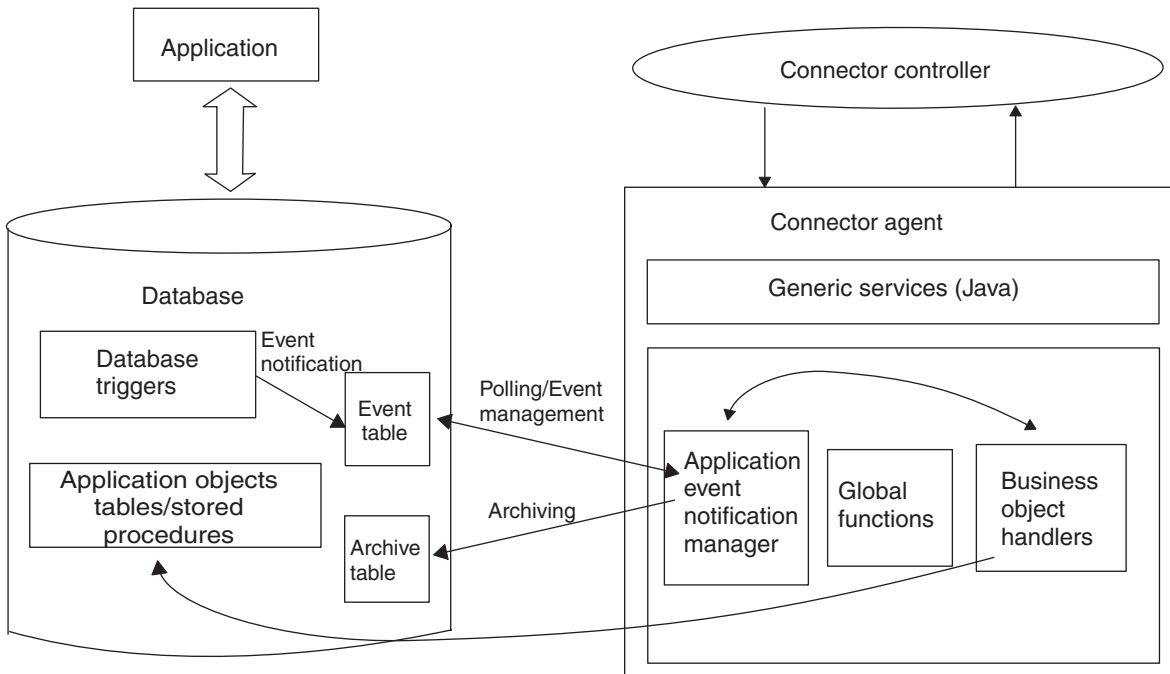


Figure 1. Business object request architecture

How the connector works

This section describes how meta-data enhances the connector’s flexibility, and presents a high-level description of business object processing and event notification.

The connector and meta-data

The connector is meta-data-driven. **Meta-data**, in the IBM WebSphere Business Integration Adapter environment, is application-specific data that is stored in business objects and that assists the connector in its interaction with the application. A meta-data-driven connector handles each business object that it supports based on meta-data encoded in the business object definition rather than on instructions hard coded in the connector.

Business object meta-data includes the structure of a business object, the settings of its attribute properties, and the content of its application-specific information. Because the connector is meta-data driven, it can handle new or modified business objects without requiring modifications to the connector code.

The connector executes SQL statements or stored procedures to retrieve or change data in the database/application. To build dynamic SQL statements or stored procedures, the connector uses application-specific meta-data. These SQL statements and stored procedures perform the required retrieval from or changes to the database/application for the business object and for the verb that the connector is processing. For information using application-specific information, see Chapter 3, “Understanding business objects for the connector,” on page 31.

Business object processing

This section provides an overview of how the connector processes business object requests and application events. For more detailed information, see “Business object verb processing” on page 36.

Processing business object requests

When the connector receives a request to perform an application operation, the connector processes hierarchical business objects recursively; that is, it performs the same steps for each child business object until it has processed all individual business objects. The order in which the connector processes child business objects and the top-level business object depends on whether the child business objects are contained with or without ownership and whether they are contained with single cardinality or multiple cardinality.

Note: The term **hierarchical** business object refers to a complete business object, including all the child business objects that it contains at any level. The term **individual** business object refers to a single business object, independent of any child business objects it might contain or that contain it. The term **top-level** business object refers to the individual business object at the top of the hierarchy that does not itself have a parent business object.

Business object retrieval: When an integration broker asks the connector to retrieve a hierarchical business object from the database, the connector attempts to return a business object that exactly matches the current database representation of that business object. In other words, all simple attributes of each individual business object returned to the integration broker match the value of the corresponding field in the database. Also, the number of individual business objects in each array contained by the returned business object match the number of children in the database for that array.

To perform such a retrieval, the connector uses the primary key values in the top-level business object to recursively descend through the corresponding data in the database.

Business object RetrievalByContent: When an integration broker asks the connector to retrieve a hierarchical business object based on values in non-key attributes in the top-level business object, the connector uses the value of all non-null attributes as the criteria for retrieving the data.

Business object creation: When an integration broker asks the connector to create a hierarchical business object in the database, the connector performs the following steps:

1. Recursively creates each single-cardinality child business object contained with ownership into the database.
2. Processes each single-cardinality child business object contained without ownership.
3. Creates the top-level business object in the database.
4. Creates each single-cardinality child business object that stores the parent/child relationship in the child.
5. Creates each multiple-cardinality child business object.

Business object modification: When an integration broker asks the connector to update a hierarchical business object in the database, the connector performs the following steps:

1. Uses the primary key values of the source business object to retrieve the corresponding entity from the database.
2. Recursively updates all single-cardinality children of the top-level business object.
3. For single-cardinality child business objects that store the relationship in the parent, sets each foreign key value in the parent to the value of the primary key in the corresponding single-cardinality child business object.
4. Updates all simple attributes of the retrieved business object except those whose corresponding attribute in the source business object contain the value CxIgnore.
5. Sets all foreign key values in each child that stores the parent/child relationship in the child (both multiple-cardinality and single-cardinality) to the primary key value of its corresponding parent business object.
6. Processes all arrays of the retrieved business object.

Business object deletion: When an integration broker asks the connector to delete a hierarchical business object from the database, the connector performs the following steps:

1. Deletes the single-cardinality children.
2. Deletes the multiple-cardinality children.
3. Deletes the top-level business object.

Processing application events

The connector handles the Create, Update, and Delete events generated by the application in the manner described below.

Create notification: When the connector encounters a Create event in the event table, it creates a business object of the type specified by the event, sets the key values for the business object (using the keys specified in the event table), and retrieves the business object from the database. After it retrieves the business object, the connector sends it with the Create verb to the integration broker.

Update notification: When the connector encounters an Update event in the event table, it creates a business object of the type specified by the event, sets the key values for the business object (using the keys specified in the event table), and retrieves the business object from the database. After it retrieves the business object, the connector sends it with the Update verb to the integration broker.

Delete notification: When the connector encounters a Delete event in the event table, it creates a business object of the type specified by the event, sets the key values for the business object (using the keys specified in the event table), and sends it with the Delete verb to the integration broker. All values other than the key values are set to CxIgnore. If any of the non-key fields are significant at your site, modify the value of the fields as needed.

The connector handles logical and physical Delete operations that are triggered by its application. In the case of physical deletes, the SmartFiltering mechanism removes all of the business object's unprocessed events (such as Create or Update) before inserting the Delete event into the event table. In the case of logical deletes, the connector inserts a Delete event in the event table without removing other events for the business object.

Retrieving business objects for event processing: A Retrieve can be done in two ways on a business object for event processing. The first is a Retrieve based on key attributes in a business object. The second is a Retrieve based on both key and

non-key attributes. In this case, the business object needs to support the RetrieveByContent verb and must use name_value pair for the object keys.

Note: If the object key does not use name_value pair, the keys in the object key field should follow the same order as the keys in the business object.

Event notification

The connector's event detection mechanism uses an event table, an archive table, stored procedures, and database triggers. Because there are potential failure points associated with the processing of events, the event management process does not delete an event from the event table until it has been inserted it into the archive table.

The database triggers populate an event table whenever an event of interest occurs in the database. The connector polls this table at a regular, configurable 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.

Note: You must add the triggers to the database as part of the installation procedure.

The setting of its ArchiveProcessed property determines whether the connector archives an event into the archive table after updating its status. For more information on the ArchiveProcessed property, see "Configuring the connector" on page 14.

Table 1 illustrates the archiving behavior depending on the setting of the ArchiveProcessed property.

Table 1. Archiving behavior

Archive processed setting	Reason deleted from event table	Connector behavior
true or no value	Successfully processed	Archived with status of Sent to InterChange
	Unsuccessfully processed	Archived with status of Error
false	No subscription for business object	Archived with status of Unsubscribed
	Successfully processed	Not archived and deleted from event table
	Unsuccessfully processed	Remains in event table with status of Error
	No subscription for business object	Remains in event table with status of Unsubscribed

SmartFiltering is a mechanism within the database triggers that minimizes the amount of processing the integration broker and connector must perform. For example, if an application has updated the Contract business object 15 times since the connector last polled for events, the SmartFiltering stores those changes as a single Update event.

Handling lost database connections

There are numerous reasons for losing a database connection. If this occurs, the connector terminates. The JDBC specification does not provide a mechanism for

detecting lost connections. As this connector supports different databases, there is no single error code definition for a lost connection to a database.

The PingQuery property is provided to handle this detection. If a failure occurs during a service call request, the connector executes this PingQuery to confirm that the failure was not due to a lost connection to a database. If the PingQuery fails and the AutoCommit property is set to false, the connector will attempt to create a new connection to the database. If it succeeds in creating a new connection to the database, it will continue processing, otherwise, the connector returns an APPRESPONSETIMEOUT, which results in the termination of the connector.

The PingQuery is executed if a failure occurs when accessing a database for any type of transaction. For example:

- While accessing the event and archive tables
- While retrieving the business object that is related to the event
- While creating or updating a record pertaining to a business object

Processing locale-dependent data

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

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

To log error and informational messages in the appropriate language and for the appropriate country or territory, configure the Locale standard configuration property for your environment. For more information on these properties, see Appendix A, “Standard configuration properties for connectors,” on page 85.

Chapter 2. Installing and configuring the connector

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

- “Adapter environment”
- “Prerequisites” on page 8
- “Installing the adapter and related files” on page 9
- “Installed file structure” on page 9
- “Enabling the application for the connector” on page 10
- “Enabling multi-driver support” on page 13
- “Enabling the custom business object handler class” on page 14
- “Configuring the connector” on page 14
- “Starting the connector” on page 28

Adapter environment

Before installing, configuring, and using the adapter, you must understand its environment requirements. They are listed in the following section.

- “Broker compatibility”
- “Adapter platforms” on page 8
- “Globalization” on page 8

Broker compatibility

The adapter framework that an adapter uses must be compatible with the version of the integration broker (or brokers) with which the adapter is communicating. Version 2.4.x of the adapter for JDBC is supported on the following adapter framework and integration brokers:

- **Adapter framework:**
WebSphere Business Integration Adapter Framework versions 2.1, 2.2, 2.3.x, and 2.4.
- **Integration brokers:**
 - WebSphere InterChange Server, versions 4.1.1, 4.2, 4.2.1, 4.2.2
 - WebSphere MQ Integrator, version 2.1.0
 - WebSphere MQ Integrator Broker, version 2.1.0
 - WebSphere Business Integration Message Broker, version 5.0
 - WebSphere Application Server Enterprise, version 5.0.2, with WebSphere Studio Application Developer Integration Edition, version 5.0.1

See *Release Notes* for any exceptions.

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

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

For WebSphere message brokers, see *Implementing Adapters with WebSphere Message Brokers*.

For WebSphere Application Server, see *Implementing Adapters with WebSphere Application Server*.

Adapter platforms

The adapter is supported on the following software.

Operating systems:

- AIX 4.3.3, AIX 5.1, AIX 5.2
- Solaris 7.0, Solaris 8.0
- Windows 2000

Databases:

Any database for which JDBC drivers are available.

Third-party software:

- JDBC drivers

Adapter dependencies

The adapter for JDBC requires the following software.

- JDBC driver files

Globalization

This adapter is DBCS (double-byte character set)-enabled and is translated.

Prerequisites

Before you use the connector, you must do the following:

- Install the Adapter Development Kit.

If the connector runs on a different machine from the integration broker, install the Adapter Development Kit that is compatible with the version of your integration broker.

- Install the JDBC driver that will be used.
- Verify that all required vendor-specific software, including JDBC driver requirements, has been installed.

For example, if you are using the JDBC Type 2 driver for an Oracle database, you must install the OracleOCI libraries.

- Verify the existence of a user account in the application.

The connector processes data in any application built on any database supported by a driver that follows the JDBC specification. For the connector to process data in the database, with which it talks directly, it must have access to a user account and password that is valid for the application. The user account must have the privileges to retrieve, insert, update, and delete data from the application's database. If you do not already have such an account, you must create one.

- Verify the character code set of the connected database.

The Java runtime environment within the Java Virtual Machine (JVM) represents data in the Unicode character code set. Unicode contains encodings for characters in most known character sets (both single-byte and multibyte). Because the connector is written in Java, it understands Unicode.

Installing the adapter and related files

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

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

Installed file structure

The following subsections describe the installed file structure of the adapter on a UNIX or Windows system.

Note: Unless otherwise indicated, the remaining sections in this chapter apply to both UNIX and Windows installations of the connector.

Installing on a UNIX system

For instructions on installing the JDBC adapter on a UNIX system, see the *Installation Guide for WebSphere Business Integration Adapters*.

Table 2 describes the UNIX file structure used by the connector.

Table 2. Installed UNIX file structure for the connector

Subdirectory of $\$ProductDir$	Description
connectors/JDBC	Contains the connector CWJDBC.jar and the start_JDBC.sh files. The start_JDBC.sh file is a system startup script for the connector. It is called from the generic connector manager script. When you click Install from Connector Configurator (WebSphere MQ Integrator Broker as the integration broker) or the Connector Configuration screen of System Manager (ICS as the integration broker), the installer creates a customized wrapper for this connector manager script. When the connector works with ICS, use this customized wrapper to start and stop the connector. When the connector works with WebSphere MQ Integrator Broker, use this customized wrapper only to start the connector. Use the <code>mqsiremotestopadapter</code> command to stop the connector.
connectors/JDBC/dependencies	Contains the SQL scripts that create the event, archive, and unique identifier tables.
connectors/messages	Contains the JDBCConnector.txt file as well as the JDBCConnector_II_TT.txt files (message files specific to a language (II) and a country or territory (TT)).
repository/JDBC	Contains the CN_JDBC.txt file.
connectors/JDBC/Samples	Contains README_Samples.txt and sample files for creating different stored procedures and business objects.
/lib	Contains the WBIA.jar file.
/bin	Contains the CWConnEnv.sh file.

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

- *System Installation Guide for UNIX* (when ICS is used as the integration broker)

- *IBM WebSphere Business Integration Adapters Implementation Guide for WebSphere MQ Integrator Broker (when WebSphere MQ Integrator Broker is used as the integration broker)*

Installed file structure on a Windows system

For instructions on installing the JDBC adapter on a Windows system, see the *Installation Guide for WebSphere Business Integration Adapters*. Table 3 describes the Windows file structure used by the connector.

Table 3. Installed Windows file structure for the connector

Subdirectory of %ProductDir%	Description
connectors\JDBC	Contains the connector CWJDBC.jar and the start_JDBC.bat files.
connectors\JDBC\dependencies	Contains the SQL scripts that create the event, archive, and unique identifier tables.
connectors\messages	Contains the JDBCConnector.txt file as well as the JDBCConnector_II_TT.txt files (message files specific to a language (II) and a country or territory (TT)).
repository\JDBC	Contains the CN_JDBC.txt file.
connectors\JDBC\Samples	Contains README_Samples.txt and sample files for creating different stored procedures and business objects.
\lib	Contains the WBIA.jar file.
\bin	Contains the CWConnEnv.bat file.

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

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

- *System Installation Guide for Windows* (when ICS is used as the integration broker)
- *IBM WebSphere Business Integration Adapters Implementation Guide for WebSphere MQ Integrator Broker* (when WebSphere MQ Integrator Broker is used as the integration broker)

Enabling the application for the connector

You must set up the event notification mechanism in the database before the connector can process event delivery. To do this, you must complete the following tasks:

- Create the event and archive tables in the database.
- Install database triggers on the application's tables to support the required business objects. It is assumed that you develop your own database triggers.
- Optionally, install a counter table. Perform this step only if you require the connector to generate a unique ID when creating a business object. For more information on generating unique IDs, see the `UID=CW.uidcolumnname[=UseIfMissing]` parameter.

The sections that follow provide information on creating and configuring the event and archive tables.

Event and archive tables

The connector uses the event table to queue events for pickup. If you have set the `ArchiveProcessed` property to true or to no value, the connector uses the archive table to store events after updating their status in the event table.

For each event, the connector gets the business object's name, verb, and key from the event table. The connector uses this information to retrieve the entire entity from the application. If the entity was changed after the event was first logged, the connector gets the initial event and all subsequent changes. In other words, if an entity is created and updated before the connector gets it from the event table, the connector gets both data changes in the single retrieval.

The following three outcomes are possible for each event processed by a connector:

- Event was processed successfully
- Event was not processed successfully
- Event was not subscribed to (for subscription information specific to your integration broker, refer to the broker's implementation guide)

If events are not deleted from the event table after the connector picks them up, they occupy unnecessary space there. However, if they are deleted, all events that are not processed are lost and you cannot audit the event processing. Therefore, it is recommended that you also create an archive table and keep the `ArchiveProcessed` property set to true. Whenever an event is deleted from the event table, the connector inserts it into the archive table.

Note: If problems accessing the application database cause the connector to fail while deleting an event from the event table or inserting an event into the archive table, the connector returns `APPRESPONSETIMEOUT`.

Configuring event and archive processing

To configure event and archive processing, you must use configuration properties to specify the following information:

- The name of the event table (`EventTableName`). You need not specify a value for this property if you use the connector only to process business object requests.
- The interval frequency ("`PollFrequency`" on page 97).
- The number of events for each polling interval (`PollQuantity`).
- The name of the archive table (`ArchiveTableName`).
- Whether the connector archives unsubscribed and unprocessed events (`ArchiveProcessed`). For subscription information specific to your integration broker, refer to the broker's implementation guide.
- The unique ID of the connector, which is important when multiple connectors poll the same table (`ConnectorID`).

You can also specify a value for the `EventOrderBy` property to specify the order of events to be processed. For information on these and other configuration properties, see Appendix A, "Standard configuration properties for connectors," on page 85 and Table 6 on page 15.

Note: Creation of the event and archive tables is optional. However, if you specify a value for `EventTableName` but do not use the connector to poll for events and do not create an event table, the connector times out. To prevent such time-out, leave the value of `EventTableName` as null (as a string).

By default, the name of the event queue table is `xworlds_events`, and the name of the archive queue table is `xworlds_archive_events`.

To use the connector only for request processing, use the `-fno` option when starting it and set the value of `EventTableName` to `null` (as a string).

If the driver being used does not support Java class `DatabaseMetaData`, and you want the connector to avoid the checking for the existence of event and archive tables, disable the `CheckForEventTableOnInit` by setting its value to `false`. By default, it is `true`. It is recommended that the value not be set to `false`.

Note: If your site will not archive events into the archive table, set the value of `ArchiveProcessed` to `false`.

SQL scripts for installing the event and archive tables

The scripts to install the event, archive, and unique identifier tables for a DB2 database are:

- `event_table_db2.sql`
- `event_package_db2.sql`
- `archive_table_db2.sql`
- `uid_table_db2.sql`

The scripts to install the event, archive, and unique identifier tables for an Oracle database are:

- `event_table_oracle.sql`
- `event_package_oracle.sql`
- `archive_table_oracle.sql`
- `uid_table_oracle.sql`

The scripts to install the event, archive, and unique identifier tables for a Microsoft SQL Server database are:

- `event_table_mssqlserver.sql`
- `event_package_mssqlserver.sql`
- `archive_table_mssqlserver.sql`
- `uid_table_mssqlserver.sql`

These files are located in the following directories:

UNIX:

`connectors/JDBC/dependencies/`

Windows:

`connectors\JDBC\dependencies\`

Note: These scripts are provided only as a template to assist you in creating the required tables for the connector. For other databases, please create your scripts using these as guidelines. The order and data type in the table columns is very important. Please refer to the “Event and archive table schema” on page 13 to view the correct order and type.

It is recommended that the DBA or person implementing the connector modify these scripts to meet specific installation and query optimization requirements. For

example, these scripts do not create indexes on the tables. It is the responsibility of the person implementing the connector to create indexes to enhance performance with the query optimizer.

Event and archive table schema

Table 4 describes the columns in the event and archive tables.

Table 4. Event and archive table schema

Name	Description	Type	Constraint
event_id	Internal identifier of the event	NUMBER	Primary key
connector_id	Unique ID of the connector for which the event is destined. This value is important when multiple connectors poll the same table.	VARCHAR	
object_key	Primary key of the business object. The key can be represented as a name_value pair, or as a set of keys delimited by a colon or other configurable delimiter (for example, 1000065:10056:2333). See the "EventKeyDel" on page 20 property for more information.	VARCHAR	Not null
object_name	Name of the business object	VARCHAR	Not null
object_verb	Verb associated with the event	VARCHAR	Not null
event_priority	Event priority (0 is highest, <i>n</i> is lowest), which the connector uses to get events on a priority basis. The connector does not use this value to lower or raise priorities.	NUMBER	Not null
event_time	Date and time the event occurred	DATETIME	Default current date/time (for archive table, actual event time)
archive_time	Date and time the event was archived (applies only to the archive table)	DATETIME	Archive date/time
event_status	-2 (Error sending event to the integration broker) -1 (Error processing event) 0 (Ready for poll) 1 (Sent to the integration broker) 2 (No Subscriptions for the business object) 3 (In Progress). This status is used only in the event table and not in the archive table.	NUMBER	Not null
event_comment	Description of the event or error string	VARCHAR	

Enabling multi-driver support

You can specify the driver by doing the following:

1. Install the driver on your machine.
2. Put all dynamic libraries that the connector requires at runtime in the connectors/JDBC directory under the product directory.
3. Edit the connector's start file to include all relevant class pathnames (including license information if required) in the JDBC_DRIVER_PATH variable.

On UNIX, the start file is:

```
$ProductDir/connectors/JDBC/start_JDBC.sh
```

On Windows, the start file is:

```
%ProductDir%\connectors\JDBC\start_JDBC.bat
```

4. Specify a value for the JDBC_DRIVER_CLASS configuration property.

Note: For all features that it supports, the connector can operate with any driver that follows the JDBC 2.0 or above specification. If the driver does not support a particular feature, the connector does not function properly. For example, if the driver does not support all method calls used by JDBCODAS, the JDBCODAS log indicates the process that the driver does not support. In such a case, you must use a different driver.

Enabling the custom business object handler class

The connector supports the custom business object handler class, CustomBOH. It implements the JDBCBOhandlerInterface interface. The syntax of this interface is:

```
public interface JDBCBOhandlerInterface{
    public int doVerbForCustom(CWConnectorBusObj busObj) throws
        VerbProcessingFailedException, ConnectionFailureException;
}
```

When you implement the doVerbForCustom method, ensure that it throws but does not catch the two exceptions. Also set the status and message of each exception before throwing them.

- **VerbProcessingFailedException**—Thrown when the operation specified by the verb fails.
- **ConnectionFailureException**—Thrown when the connector can not establish a connection with the application.

To enable the connector to support this business object handler:

- Specify the CustomBOH class name in the verb application-specific information. The connector obtains the name of the custom business object handler class from the verb application-specific information. Use the following syntax:

```
CustomBOH=customBOhandlerClassName
```

For example, assume the verb application-specific information is specified as follows:

```
CustomBOH=JDBCBOhandlerForOverrideSQL
```

In this case, JDBCBOhandlerForOverrideSQL is the name of the custom business object handler class.

- Ensure that CustomBOH belongs to com.crossworlds.connectors.JDBC. If the connector finds "CustomBOH=" in the verb application-specific information and finds the class in the com.crossworlds.connectors.JDBC package, it executes the custom business object handler. If it does not find CustomBOH, it throws an error saying that it could not find the class.

Configuring the connector

You must set the connector's standard and connector-specific configuration properties before you can run it. Use one of the following tools to set a connector's configuration properties:

- **Connector Configurator (if ICS is the integration broker)**--Access this tool from the System Manager.
- **Connector Configurator (if WebSphere MQ Integrator Broker is the integration broker)**--Access this tool from the IBM WebSphere Business Integration Adapter program folder. For more information about Connector Configurator, see Appendix B, "Connector Configurator," on page 101.

Standard connector properties

Standard configuration properties provide information that all connectors use. See Appendix A, “Standard configuration properties for connectors,” on page 85 for detailed information about these properties.

Important: Because the connector for JDBC supports both the ICS and WebSphere MQ Integrator Broker integration brokers, configuration properties for both brokers are relevant to the connector.

In addition, refer to Table 5 for configuration information specific to the IBM WebSphere Business Integration Adapter for JDBC. The information in this table supplements the information in the appendix.

Table 5. Property information specific to this connector

Property	Notes
CharacterEncoding	This connector does not use the CharacterEncoding property
Locale	Because this connector has been internationalized, you can change the value of the Locale property. Note: If you are using WebSphere MQ Integrator Broker as your broker, you must use the same locale for the adapter, the broker, and any applications.

Note that you must provide a value for the ApplicationName configuration property before running the connector.

Connector-specific properties

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

Table 6 lists the connector-specific configuration properties for the connector. See the sections that follow for explanations of the properties.

Table 6. Connector-specific configuration properties

Name	Possible values	Default value	Required
ApplicationPassword	Password for connector user account		Yes*
ApplicationUserName	Name of connector user account		Yes*
ArchiveProcessed	true or false	true	No
ArchiveTableName	Name of archive queue table	xworlds_archive_events	Yes if Archive Processed is true
AutoCommit	true or false	false	No
CheckforEventTableInInit	true or false	true	No
ChildUpdatePhyDelete	true or false	false	No
CloseDBConnection	true or false	false	No
ConnectorID	Unique ID for the connector	null	No
DatabaseURL	Name of the database server		Yes

Table 6. Connector-specific configuration properties (continued)

Name	Possible values	Default value	Required
DateFormat	A time pattern String	MM/dd/yyyy HH:mm:ss	No
DriverConnectionProperties	Additional JDBC driver connection properties		No
EventKeyDel	Delimiter character or characters for object key column of event table	semicolon (;)	No
EventOrderBy	none, ColumnName, ColumnName, ...]		No
EventQueryType	Fixed or Dynamic	Fixed	No
EventTableName	Name of event queue table	xworlds_events	Yes, if polling is required; null (as a string) if polling is not required
JDBCDriverClass	driver classname		Yes
MaximumDatabaseConnections	Number of simultaneous database connections	5	Yes
PingQuery	SELECT 1 FROM <tablename>		No
PollQuantity	Values are 1 to 500	1	No
PreserveUIDSeq	true or false	true	No
RDBMS.initsession	SQL statement that initializes every database session		No
RDBMSVendor	MSSQLServer, Oracle, Others		Yes
ReplaceAllStr	true or false	false	No
ReplaceStrList	A set composed of a single character, a character delimiter, and the character's substitution string. Also, multiple such sets with a termination delimiter between them.	Q,DSQNote: In the connector configuration tool, these characters represent a single quotation mark, followed by a comma, followed by two single quotation marks.	No
RetryCountAndInterval	Count, interval in seconds	3,20	No
SchemaName	Schema on which the events reside		No
SPBeforePollCall	Name of the stored procedure to be executed for each poll call		No
StrDelimiter	The character and termination delimiters used in the ReplaceStrList property	,:	No
TimingStats	0, 1, 2	0	No
UniqueIDTableName	Name of table used for generation of IDs	xworlds_uid	No
UseDefaults	true or false	false	Yes
UseDefaultsForCreatingChildBOs	true or false	false	No
UseDefaultsForRetrieve	true or false	false	No

*ApplicationPassword and ApplicationUserNames are not required if you are using trusted authentication.

ApplicationPassword

Password for the connector's user account.

There is no default value.

ApplicationUserName

Name of the connector's user account.

There is no default value.

ArchiveProcessed

Specifies whether the connector archives events for which there are no current subscriptions.

Set this property to true to cause events to be inserted into the archive table after they are deleted from the event table.

Set this property to false to cause the connector not to perform archive processing. In this case, it does not check the value of the ArchiveTableName property. If ArchiveProcessed is set to false, the connector performs the following behavior:

- If the event is successfully processed, the connector deletes it from the event table and does not archive it.
- If the connector does not subscribe to the event's business object, the connector leaves the event in the event table and changes its event status to Unsubscribed. For subscription information specific to your integration broker, refer to the broker's implementation guide.
- If the business object encounters a problem while being processed, the connector leaves the event in the event table with event status of Error.

If this property is set to false and the poll quantity is low, the connector appears to be polling the event table, but it is simply picking up the same events repeatedly.

If this property has no value, the connector assumes the value to be true. If the ArchiveTableName property also has no value, the connector assumes the archive table's name is `xworlds_archive_events`.

The default value is true.

ArchiveTableName

Name of archive queue table.

If the ArchiveProcessed property is set to false, it is unnecessary to set a value for this property.

The default name is `xworlds_archive_events`.

AutoCommit

This property makes the AutoCommit setting configurable. When set to true, all transactions are automatically committed. Some databases (such as Sybase) require AutoCommit to be set to true. If set to false, stored procedures on Sybase to fail.

If the database connection is lost, the connector will attempt to create a new connection to restart the complete processing as long as AutoCommit is set to false. If the new connection is invalid, or if AutoCommit is set to true, the connector returns `APPRESPONSETIMEOUT`, which results in the termination of the connector.

The default value is false.

CheckforEventTableInInit

Setting this connector property to false prevents the connector from checking for the existence of the event and archive tables during connector initialization. It is recommended that you always set it to true unless the JDBC driver you are using does not support the JDBC class DatabaseMetaData.

When the property is set to false, although the connector does not check for the existence of EventTable and ArchiveTable, the event and archive tables should always exist because the connector uses them during the initialization process. To prevent the connector from using the event and archive tables during initialization, set the property EventTableName to null.

The default value is true.

ChildUpdatePhyDelete

During an update operation, specifies how the connector handles data represented by a child business object that is missing from the incoming business object but exists in the database.

Set this property to true to cause the connector to physically delete the data record from the database.

Set this property to false to cause the connector to logically delete the data record from the database by setting the status column to the appropriate value. The application-specific information obtains the name of the status column and its value from the StatusColumnValue (SCN) parameter specified in its business-object level application-specific information. For more information, see “Application-specific information at the business-object level” on page 54.

Default value is false.

CloseDBConnection

This property makes the closing of the database connection configurable. When set to true, for every service call request and poll call, the database connection is closed. Setting this property to true impairs performance and is not advisable.

The default value is false.

ConnectorID

A unique ID for the connector. This ID is useful to retrieve events for a particular instance of the connector.

Default value is null.

DatabaseURL

Name of the database server to which the connector should connect.

If you use the WebSphere business integration system branded SQLServer driver, the recommended URL is:

```
jdbc:ibm-crossworlds:sqlserver://MachineName:PortNumber;DatabaseName=DBname
```


Important

If `AutoCommit` is set to `false`, you must set an additional parameter, `SelectMethod: jdbc:ibm-crossworlds:sqlserver://MachineName:PortNumber;DatabaseName=DBname; SelectMethod=cursorBy default, SelectMethod is set to direct. For more information, see "AutoCommit" on page 17.`

You must provide a value for this property in order for the connector to process successfully.

DateFormat

Specifies the date format that the connector expects to receive and return. This property supports any format that is based on the syntax as contained in Table 7.

Table 7 Defines the time format syntax using a time pattern string. In this pattern, all ASCII letters are reserved as pattern letters.

Table 7. Time format syntax

Symbol	Meaning	Presentation	Example
G	era designator	(Text)	AD
y	year	(Number)	1996
M	month in year	(Text & Number)	July & 07
d	day in month	(Number)	10
h	hour in am/pm(1-12)	(Number)	12
H	hour in day(0-23)	(Number)	0
m	minute in hour	(Number)	30
s	second in minute	(Number)	55
S	millisecond	(Number)	978
E	day in week	(Text)	Tuesday
D	day in year	(Number)	189
F	day of week in month	(Number)	2 (2nd Wed in July)
w	week in year	(Number)	27
W	week in month	(Number)	2
a	am/pm marker	(Text)	PM
k	hour in day(1-24)	(Number)	24
K	hour in am/pm(0-11)	(Number)	0
z	time zone	(Text)	Pacific Standard Time
'	escape for text	(Delimiter)	
''	single quote	(Literal)	'

Table 8. Examples using the US locale

Format pattern	Result
"yyyy.MM.dd G 'at' hh:mm:ss z"	1996.07.10 AD at 15:08:56 PDT

Table 8. Examples using the US locale (continued)

Format pattern	Result
"EEE, MMM d, ''yy"	Wed, July 10, '96
"h:mm a"	12:08 PM
"hh 'o'clock' a, zzzz"	12 o'clock PM, Pacific Daylight Time
"K:mm a, z"	0:00 PM, PST
"yyyy.MMMM.dd GGG hh:mm aaa"	1996.July.10 AD 12:08 PM

DriverConnectionProperties

Besides the user name and password, a JDBC driver might need additional properties or information. The `DriverConnectionProperties` connector property will take additional properties that a JDBC driver needs, as name-value pairs. The properties should be specified as follows:

```
property1=value1[:property2=value2...]
```

The properties must be given as name value pairs, separated by semi-colons. The property is separated from its value by an equals sign (with no extra spaces).

For example, assume the JDBC driver needs license information and port number. The property name it expects for license information is `MyLicense` and the value is `ab23jk5`. The property name it expects for port number is `PortNumber` and value is `1200`. The `DriverConnectionProperties` should be set to the value `MyLicense=ab23jk5;PortNumber=1200`.

EventKeyDel

Specifies the delimiter when the `object_key` column of the event table contains multiple attribute values.

There are two ways to retrieve the business object that has been created, updated, or deleted in the triggering application.

- The first is to populate the `object_key` column with values for attributes that are keys in a business object. Set the `EventKeyDel` configuration property to a single character that is not part of the key field. For **example**, if the delimiter is specified as `“;”`, then the `object_key` will be as follows: `xxx;123`
- The second is to populate the `object_key` column with values for any attribute in a business object. These values should be represented as `name_value` pair. The first delimiter will be for the `name_value` and the second is for the keys. For example, if the delimiter is specified as `“=”`, then the `object_key` will be as follows: `CustomerName=xxx;CustomerId=123;`

If the delimiter is specified as `“=”`, then the `object_key` will be as follows:
`CustomerName=xxx:CustomerId=123:`

Note: The order that the key values are defined should follow the same order as the key attributes in a business object.

Important: If you use Date attribute data, avoid using a colon (:) delimiter, because it may be included in the attribute's data.

The default value is a semicolon (;), which is based on keys, not `name_value` pairs.

EventOrderBy

Specifies whether to turn off the ordering of events, or specifies an order of event processing that is different from the default order.

By default, at each poll the connector pulls only the number of events specified in its `PollQuantity` property, and orders event processing by the values in the `event_time` and `event_priority` columns of the Event table.

To cause the connector not to order events, set the value of this property to `none`.

To cause the connector to order by different columns in the Event table, specify the names of those columns. Separate column names with a comma (,). Specifying a value for this property overwrites the default value.

There is no default value for this property.

EventQueryType

The `EventQueryType` property is used to indicate whether the connector should dynamically generate a query to retrieve events from the event table or use its built in query. For the dynamically generated query, the connector maps its event structure to the columns in the event table. The order of the data in the table columns is very important. Please refer to the “Event and archive table schema” on page 13 to view the correct order.

If the value in the `EventQueryType` is `Fixed` (as a string), the default query is executed. If set to `Dynamic` (as a string), a new query is built by getting the column names from the table that is specified in the “`EventTableName`” property.

The event table column names can change but the order and data type of the columns must remain the same as specified in the event table creation section. “`EventOrderBy`” on page 20 will be appended to either the default or the dynamically generated query.

If the `EventQueryType` property is not added or it contains no value, it is defaulted to `Fixed`.

Default value is `Fixed` (as a string).

EventTableName

Name of event queue table, which is used by the connector’s polling mechanism.

The default name is `xworlds_events`.

Set this to `null` (as a string) when polling is turned off for the connector. This prevents validation of the existence of the event and archive tables.

For a user defined event table, ensure that the `event_id` maps to one of the following JDBC types: `INTEGER`, `BIGINT`, `NUMERIC`, `VARCHAR`.

JDBCDriverClass

Specifies the class name of a driver. To use a particular JDBC driver, specify the driver’s class name in this configuration property. For example, to specify the Oracle thin driver, set the value of this property to:
`oracle.jdbc.driver.OracleDriver`.

For more information, see “Enabling multi-driver support” on page 13 and “UseDefaultsForCreatingChildBOs” on page 26.

No default value is provided.

MaximumDatabaseConnections

Specifies the maximum number of simultaneous database connections allowed. At runtime, the number of open database connections is the sum of this value plus 1.

If the “PreserveUIDSeq” property is set to false, at runtime, the number of open database connections is the sum of this value plus 2.

The default value is 5.

PingQuery

Specifies the SQL statement or stored procedure that the connector executes to check database connectivity.

The following is an example of an SQL statement used as a ping query:

```
SELECT 1 FROM <tablename>
```

The following is an example of a stored procedure call (sampleSP) used as a ping query with an Oracle or DB2 database:

```
call sampleSP( )
```

Note that stored procedure calls cannot have output parameters. If an input parameter is required by the database, the input value must be specified as part of the ping query. For example:

```
Call checkproc(2)
```

There is no default value. For more information, see “Handling lost database connections” on page 5 and “Loss of connection to the application” on page 82.

PollQuantity

Number of rows in the database table that the connector retrieves per polling interval. Allowable values are 1 to 500.

The default value is 1.

PreserveUIDSeq

Specifies whether or not the incoming unique ID sequence will be preserved in the unique identifier table.

If set to true, the unique ID is not committed until the business object is successfully processed in the destination application. All other processes attempting to access the unique identifier table must wait until the transaction is committed.

If set to false, the unique ID is committed when the business object requests it. The business object processing and the unique ID processing each have their own transaction block (internal to the connector). This is only possible if the transaction relating to the unique identifier table has its own connection.

Note: If this property is not added to the connector configuration, the default behavior is the same as if this property were added and set to true. Also, if “AutoCommit” on page 17 is set to true, the connector executes the same behavior as if PreserveUIDSeq is set to false.

If the “PreserveUIDSeq” property is set to false, at runtime, the number of open database connections is the sum of this value plus 2.

The default value is true.

RDBMS.initsession

SQL statement that initializes every session with the database. The connector takes a query and executes it at startup. There should not be a return value for this query. The property name is required, but a value is not.

There is no default value.

RDBMSVendor

Specifies which RDBMS the connector uses for special processing. Set the value of this property to Oracle or MSSQLServer if you are using an Oracle or Microsoft SQL Server database. If you are using a different database, set the value to the name of that database, or leave the value blank.

If using a non-default database, ensure that the proper driver is loaded. If this property is set to Others, the connector determines which database to use by locating the driver.

A value is required for the connector to process successfully.

No default value is provided.

ReplaceAllStr

Specifies whether the connector replaces all instances of each character identified in the ReplaceStrList property with the substitution string specified in that property. The connector evaluates ReplaceAllStr only if the ESC=[true|false] parameter of each attribute's AppSpecificInfo property does not contain a value. In other words, if the ESC parameter has been specified, its value takes precedence over the value set for the ReplaceAllStr property. To cause the connector to use the value of ReplaceAllStr, verify that the ESC parameter has not been specified.

The default value of ReplaceAllStr is false.

Note: The ESC parameter and the ReplaceAllStr and ReplaceStrList properties provide support for database escape character functionality (for example, escaping single quotes). Because the same functionality is also available from the Prepared Statements provided by the JDBC driver, these properties will be deprecated in future releases of the connector. The connector currently supports the use of the JDBC Prepared Statements.

ReplaceStrList

Specifies one or more substitution sets, each composed of an individual character to be replaced, a character delimiter, and a substitution string. The connector performs this substitution on an attribute's value only if a value has been specified for the ESC=[true|false] parameter of the attribute's AppSpecificInfo property or for the connector's ReplaceAllStr property.

Note: The ESC parameter and the ReplaceAllStr and ReplaceStrList properties provide support for database escape character functionality (for example, escaping single quotes). Because the same functionality is also available from the Prepared Statements provided by the JDBC driver, these properties will be deprecated in future releases of the connector. The connector currently supports the use of the JDBC Prepared Statements.

The syntax for this attribute is:

```
single_char1,substitution_str1[:single_char2,substitution_str2[:...]]
```

where:

<i>single_char</i>	A character to be replaced.
<i>single_char</i>	The substitution string that the connector uses to replace the character.
<i>single_char</i>	The character delimiter, which separates the character to be replaced from the string that replaces it. By default, the character delimiter is a comma (.). You can configure this delimiter by setting the first delimiter in the <code>StrDelimiter</code> property.
<i>single_char</i>	The termination delimiter, which separates substitution sets (each of which is composed of the character to be replaced, a character delimiter, and the substitution string). By default, the termination delimiter is a colon (:). You can configure this delimiter by setting the second delimiter in the <code>StrDelimiter</code> property.

For example, assume you want to replace a single percent sign (%) with two percent signs (%%), and a caret (^) with a backslash and a caret (\^). By default, `StrDelimiter` specifies a comma (,) as the character delimiter, and a colon (:) as the termination delimiter. If you keep the default delimiters, use the following string as the value of `ReplaceStrList`:

```
%,%:^\,^
```

Note: A restriction of the connector configuration tool prevents entering single quotation marks. Therefore, you must represent a single quotation with the character Q, and two single quotations with the characters DSQ. In the above example, if you also want to substitute a single quotation mark (') with two single quotation marks (' '), use the following notation: Q,DSQ:%,%:^\,^

RetryCountAndInterval

Specifies the number of attempts and the interval in seconds that the connector should use when it is unable to lock data while performing an update operation.

Before it performs an update, the connector locks rows related to the update and attempts to retrieve current data. If the connector cannot lock the rows, it tries again to get the lock for the count and interval specified in this configuration property. The connector eventually times out if the lock is not obtainable within the values specified here.

Specify the value in the format: count, interval in seconds. For example, a value of 3,20 specifies three retries with an interval of 20 seconds in between.

The default is 3,20.

SchemaName

This property limits the search for the event and archive tables within that particular schema. If this property is not added or if it is left empty, the connector will search all of the schemas that the user has access to. This `SchemaName` is also used when building the queries to access the event and archive tables.

The Oracle Database provides support for schema names. For MSSQL server or DB2, a schema name can refer to a database owner name; however, refer to the respective JDBC driver documentation for specific information.

Note: DB2 schema names are case sensitive. You must specify the schema name in uppercase letters.

No default value is provided.

SPBeforePollCall

This property names the stored procedure that is executed for every poll call. If the property `SPBeforePollCall` has a value (the name of a stored procedure), then at the start of each poll call, the connector calls the stored procedure, passing it the values of the connector properties `ConnectorID` and `PollQuantity`. The procedure will update `PollQuantity` number of rows, setting the connector-id column to `ConnectorID` where `status=0` and connector-id is null. This enables load balancing in the connector.

Note: In the case where a poll call fails prematurely (the database is down, or the connection is lost), the connector-id remains set. This may result in records being skipped during polling. It is therefore recommended that periodically, the connector-id is reset back to null for all records in the event table with a status of 0.

StrDelimiter

Specifies the character and termination delimiters for use in the `ReplaceStrList` property.

- The character delimiter separates the character to be replaced from the string that replaces it. The character delimiter occupies the first (left-hand) position of this property's values and defaults to a comma (,).
- The termination delimiter separates substitution sets (each of which is composed of the character to be replaced, a character delimiter, and the substitution string). The termination delimiter occupies the second (right-hand) position of this property's values and defaults to a colon (:).

You can specify your own value for either or both of these delimiters. If you do so, do not specify a space or other character between them.

Default value is a comma followed immediately by a colon (,:)

TimingStats

Allows you to time each verb operation of the connector to look for problems. Available settings are:

- 0 (No timing statistics)
- 1 (Timing displayed at entry and exit of the verb operation for an entire hierarchical business object).
- 2 (Timing displayed at entry and exit of each verb operation for each individual business object in a hierarchical business object).

Timing messages are log messages rather than trace messages. They can be turned on and off, independent of trace levels.

The default value is 0.

UniqueIDTableName

Specifies the table that contains the latest value used for generation of a unique ID. By default, the table has one column (id). You can customize the table to add a column for each attribute that requires generation of a UID.

The default value is `xworlds_uid`.

UseDefaults

If `UseDefaults` is set to `true` or is not set, the connector checks whether a valid value or a default value is provided for each required business object attribute. If a value is provided, the `Create` succeeds; otherwise, it fails.

If `UseDefaults` is set to `false`, the connector checks only whether a valid value is provided for each required business object attribute; the `Create` operation fails if a valid value is not provided.

The default value is `false`.

UseDefaultsForCreatingChildBOs

If `UseDefaultsForCreatingChildBOs` is set to `true` or is not set, the connector checks whether a valid value or a default value is provided for each required business object attribute. If a value is provided, the `Create` succeeds; otherwise, it fails.

If `UseDefaultsForCreatingChildBOs` is set to `false`, the connector checks only whether a valid value is provided for each required business object attribute; the `Create` operation fails if a valid value is not provided.

UseDefaultsForRetrieve

For polling: If `UseDefaultsForRetrieve` is not defined and set to `true`, the default values will be set in the BO before it is retrieved from the database and dispatched to the server.

If `UseDefaultsForRetrieve` is defined and set to `false`, the default values will not be set in the BO before it is retrieved from the database and dispatched to the server.

For request processing: If `UseDefaultsForRetrieve` is not defined and set to `false`, the default values will not be set in the BO before it is retrieved from the database and dispatched to the server.

If `UseDefaultsForRetrieve` is defined and set to `true`, the default values will be set in the BO before it is retrieved from the database and dispatched to the server.

Creating multiple instances of the connector

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

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

Create a new directory

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

```
ProductDir\connectors\connectorInstance
```

where `connectorInstance` uniquely identifies the connector instance.

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

```
ProductDir\repository\connectorInstance
```

Create business object definitions

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

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

```
ProductDir\repository\initialConnectorInstance
```

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

Create a connector definition

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

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

Create a start-up script

To create a startup script:

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

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

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

Starting the connector

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

```
ProductDir\connectors\connName
```

where *connName* identifies the connector. The name of the startup script depends on the operating-system platform, as Table 9 shows.

Table 9. Startup scripts for a connector

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

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

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

- From the command line

- On Windows systems:

```
start_ connName connName brokerName [-cconfigFile ]
```

- On UNIX-based systems:

```
connector_manager_ connName -start
```

where *connName* is the name of the connector and *brokerName* identifies your integration broker, as follows:

- For WebSphere InterChange Server, specify for *brokerName* the name of the ICS instance.
- For WebSphere message brokers (WebSphere MQ Integrator, WebSphere MQ Integrator Broker, or WebSphere Business Integration Message Broker) or WebSphere Application Server, specify for *brokerName* a string that identifies the broker.

Note: For a WebSphere message broker or WebSphere Application Server on a Windows system, you *must* include the *-c* option followed by the name of the connector configuration file. For ICS, the *-c* is optional.

- From Adapter Monitor (WebSphere Business Integration Adapters product only), which is launched when you start System Manager
You can load, activate, deactivate, pause, shutdown or delete a connector using this tool.
- From System Monitor (WebSphere InterChange Server product only)
You can load, activate, deactivate, pause, shutdown or delete a connector using this tool.
- On Windows systems, you can configure the connector to start as a Windows service. In this case, the connector starts when the Windows system boots (for an Auto service) or when you start the service through the Windows Services window (for a Manual service).

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

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

Stopping the connector

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

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

where *connName* is the name of the connector.

- From Adapter Monitor (WebSphere Business Integration Adapters product only), which is launched when you start System Manager
You can load, activate, deactivate, pause, shutdown or delete a connector using this tool.
- From System Monitor (WebSphere InterChange Server product only)
You can load, activate, deactivate, pause, shutdown or delete a connector using this tool.
- On Windows systems, you can configure the connector to start as a Windows service. In this case, the connector stops when the Windows system shuts down.

Chapter 3. Understanding business objects for the connector

This chapter describes how the connector for JDBC processes business objects and describes the assumptions the connector makes when retrieving and modifying data. It contains the following sections:

- “Business object and attribute naming conventions”
- “Business object structure”
- “Business object verb processing” on page 36
- “Business object attribute properties” on page 50
- “Business object application-specific information” on page 53

You can use this information as a guide to modifying existing business objects or as suggestions for implementing new ones. For information on the utility that automates the creation of WebSphere Business Integration Adapter business object definition files from database tables, see Chapter 4, “Generating business object definitions using JDBCODA,” on page 65.

The connector makes assumptions about the structure of its supported business objects, the relationships between parent and child business objects, the format of the application-specific information, and the database representation of the business object. Therefore, when you create or modify a business object that will be processed by the connector, your modifications must conform to the rules the connector is designed to follow. If they do not, the connector cannot process new or modified business objects correctly.

Business object and attribute naming conventions

The name of a business object used by the connector can consist only of alphanumeric characters or the underscore character. Business object attribute names also can consist only of alphanumeric characters or the underscore character.

Business object structure

In most cases, the connector assumes that every individual business object is represented by one database table or view, and that each **simple attribute** (that is, an attribute that represents a single value, such as a String or Integer or Date) within the object is represented by a column in that table or view. Thus, attributes within the same individual business object cannot be stored in different database tables. However, the following situations are possible:

- The database table might have more columns than the corresponding individual business object has simple attributes (that is, some columns in the database are not represented in the business object). Include in your design only those columns needed for the business object processing.
- The individual business object might have more simple attributes than the corresponding database table has columns (that is, some attributes in the business object are not represented in the database). The attributes that do not have a representation in the database either have no application-specific information or are set with a default value or specify stored procedures.
- The individual business object can represent a view that spans multiple database tables. The connector can use such a business object when processing Create,

Retrieve, Update, and Delete events triggered in the application. However, when processing business object requests, the connector can use such a business object only for Retrieve requests.

- The individual business object can represent a wrapper object that is used as a container for unrelated business objects. The wrapper object is not represented by a database table or view. Wrapper objects may not be used as children of other objects.

Note: If a business object is based on a stored procedure, each simple attribute (other than the special SP attributes) may or may not have application-specific information. For more information, see “Stored procedures” on page 44.

WebSphere Business Integration Adapter business objects can be flat or hierarchical. All the attributes of a **flat** business object are simple and represent a single value.

A hierarchical business object has attributes that represent a child business object, an array of child business objects, or a combination of both. In turn, each child business object can contain a child business object or an array of business objects, and so on. A **single-cardinality relationship** occurs when an attribute in a parent business object represents a single child business object. In this case, the attribute is of the same type as the child business object.

A **multiple-cardinality relationship** occurs when an attribute in the parent business object represents an array of child business objects. In this case, the attribute is an array of the same type as the child business objects.

Note: The term **hierarchical** business object refers to a complete business object, including all the child business objects that it contains at any level. The term **individual** business object refers to a single business object, independent of any child business objects it might contain or that contain it. The term **top-level** business object refers to the individual business object at the top of the hierarchy that does not itself have a parent business object.

The connector supports the following relationships among business objects:

- “Single-cardinality relationships”
- “Single-cardinality relationships and data without ownership” on page 33
- “Multiple-cardinality relationships” on page 34
- “Single-cardinality relationships that store the relationship in the child” on page 35
- “Wrapper objects” on page 36

In each type of cardinality, the relationship between the parent and child business objects is described by the application-specific information of the key attribute of the business object storing the relationship. For more information on this application-specific information, see "FK=[fk_object_name.]fk_attribute_name" on page 55 in Table 11 on page 55.

Single-cardinality relationships

Typically, a business object that contains a single-cardinality child business object has at least two attributes that represent the relationship. The type of one attribute is the same as the child’s type. The other attribute is a simple attribute that

contains the child's primary key as a foreign key in the parent. The parent has as many foreign-key attributes as the child has primary-key attributes.

Because the foreign keys that establish the relationship are stored in the parent, each parent can contain only one single-cardinality child of a given type.

Figure 2 illustrates a typical single-cardinality relationship. In the example, `fk1` is the simple attribute that contains the child's primary key, and `child[1]` is the attribute that represents the child business object.

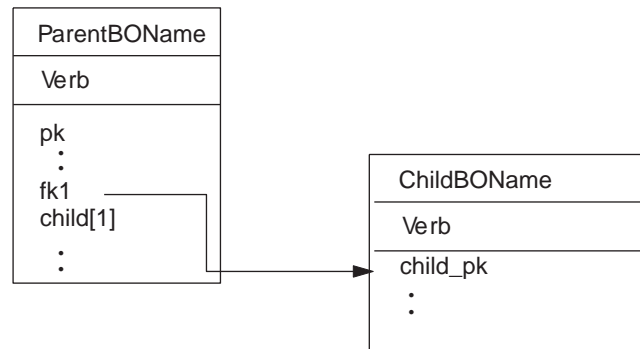


Figure 2. Typical single-cardinality relationship

Single-cardinality relationships and data without ownership

Typically, each parent business object **owns** the data within the child business object that it contains. For example, if each Customer business object contains a single Address business object, when a new customer is created, a new row is inserted into both the customer and the address tables. The new address is unique to the new customer. Likewise, when deleting a customer from the customer table, the customer's address is also deleted from the address table.

However, there are situations where multiple hierarchical business objects contain the same data, which none of them owns. For example, assume that an Address business object has a `StateProvince[1]` attribute that represents the `StateProvince` lookup table with single cardinality. Because the lookup table is rarely updated and is maintained independently of the address data, creation or modification of address data does not affect the data in the lookup table. The connector either finds an existing state or province name or fails. It does not add or change values in the lookup table.

When multiple business objects contain the same single-cardinality child business object, the foreign-key attribute in each parent business object must specify the relationship as `NO_OWNERSHIP`. When an integration broker sends the connector a hierarchical business object with a Create, Delete, or Update request, the connector ignores single-cardinality children contained without ownership. The connector performs only retrieves on these business objects. If the connector fails to retrieve such a single-cardinality business object, it returns an error and stops processing.

For information on how to specify the relationship without ownership, see "Attributes that represent a single-cardinality child business object" on page 61. For more information on specifying foreign key relationships, see "Specifying an attribute's foreign key" on page 57.

Denormalized data and data without ownership

In addition to facilitating use of static lookup tables, containment without ownership provides another capability: synchronizing normalized and denormalized data.

Synchronizing from normalized to denormalized data: Specifying a relationship as `NO_OWNERSHIP` allows you to create or change data when you synchronize from a normalized application to a denormalized one. For example, assume that your normalized source application stores data in two tables, A and B. Assume further that your denormalized destination application stores all the data in a single table such that each entity A redundantly stores B data.

In this example, to synchronize a change in table B data from your source application to your destination application, you must trigger a table A event whenever table B data changes. Moreover, because table B data is stored redundantly in table A, you must send a business object for each row in table A that contains the changed data from table B.

Synchronizing from denormalized to normalized data: When synchronizing data from a denormalized source application to a normalized destination application, the connector does not create, delete, or update data contained without ownership in the normalized application.

When synchronizing data to a normalized application, the connector ignores all single-cardinality children contained without ownership. In order to create, remove, or modify such child data, you must process the data manually.

Multiple-cardinality relationships

Typically, a business object that contains an array of child business objects has only one attribute that represents the relationship. The type of the attribute is an array of the same type as the child business objects. In order for a parent to contain more than one child, the foreign keys that establish the relationship are stored in each child.

Therefore, each child has at least one simple attribute that contains the parent's primary key as a foreign key. The child has as many foreign-key attributes as the parent has primary key attributes.

Because the foreign keys that establish the relationship are stored in the child, each parent can have zero or more children.

Figure 3 illustrates a multiple-cardinality relationship. In the example, `parentId` is the simple attribute that contains the parent's primary key, and `child[n]` is the attribute that represents the array of child business objects.

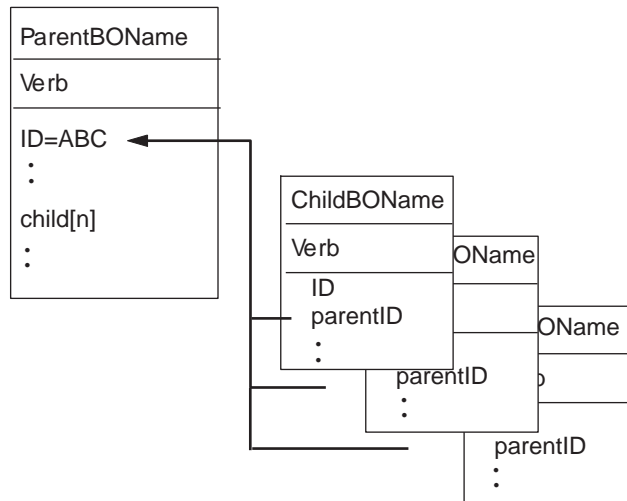


Figure 3. Multiple-cardinality business object relationship

Single-cardinality relationships that store the relationship in the child

Some applications store a single child entity so that the relationship is stored in the child rather than in the parent. In other words, the child contains a foreign key whose value is identical to the value stored in the parent's primary key.

Figure 4 illustrates this special type of single-cardinality relationship.

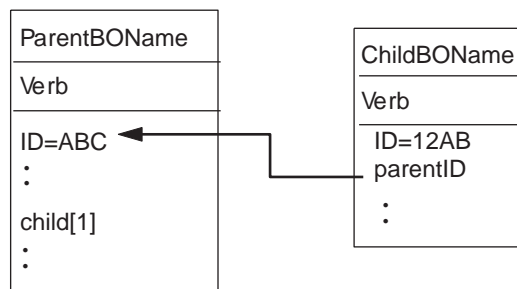


Figure 4. Single-cardinality business object with relationship stored in the child

Applications use this type of single-cardinality relationship when child data does not exist independently of its parent and can be accessed only through its parent. Such child data is never owned by more than one parent, and requires that the parent and its primary-key value exist before the child and its foreign-key value can be created.

To accommodate such applications, the connector also supports hierarchical business objects that contain a child with single cardinality but store the relationship in the child rather than in the parent.

To specify that a parent business object contains a single-cardinality child in this special way, when you specify the application-specific information of the attribute

that contains the child, do not include the `CONTAINMENT` parameter. For more information, see “Attributes that represent a single-cardinality child business object” on page 61.

Wrapper objects

The wrapper object is a top-level business object that does not correspond to any database table or view. The wrapper object is denoted by the top-level business object property of `WRAPPER` with a value of `true`. The wrapper object is a dummy parent that is used as a container for unrelated children. In processing the wrapper object, the connector ignores the top-level business object and processes only the children. The wrapper object may contain `N` cardinality or `N-1` cardinality entities or both.

A `N` cardinality entity should have at least one unique attribute marked as a primary key and at least one attribute marked as a foreign key. This foreign key will then be added as a primary key in the wrapper object. The entity’s foreign key will reference the wrapper object’s primary key that was just added.

In the case of a `N-1` cardinality entity, the primary key should be marked as both a primary key and a foreign key, referencing the primary key in the wrapper, which is the same as the primary key in the `N-1` entity.

Business object verb processing

This section describes the following aspects of processing a business object’s verbs:

- “Verb determination,” which explains how the connector determines the verb to use for each individual, source business object
- “Afterimages and deltas,” which defines the terms and explains how the connector works with afterimages
- “Verb processing” on page 38, which explains the steps the connector takes when creating, retrieving, updating, or deleting a business object
- “SQL statements” on page 44, which explains how the connector uses simple SQL statements for selecting, updating, retrieving or deleting operations
- “Stored procedures” on page 44, which explains how the connector uses stored procedures
- “Transaction commit and rollback” on page 50, which briefly explains how the connector uses transaction blocks

Verb determination

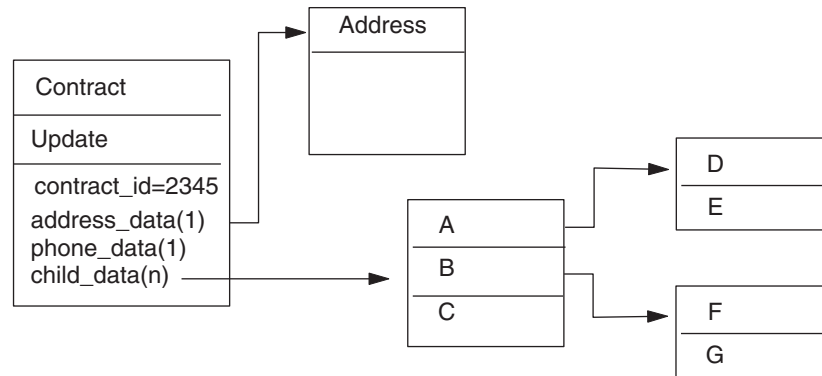
A top-level business object and each of its individual child business objects can contain their own verbs. Therefore, an integration broker can pass a business object that has different verbs for parent and child business objects to the connector. When this occurs, the connector uses the verb of the top-level parent business object to determine how to process the entire business object. For more information, see “Verb processing” on page 38.

Afterimages and deltas

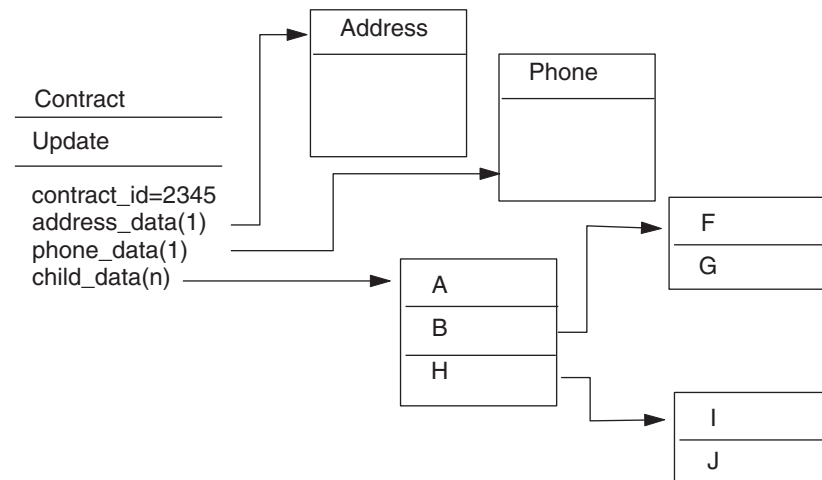
An afterimage is the state of a business object after all changes have been made to it. A delta is a business object used in an update operation that contains only key values and the data to be changed. Because the connector supports only afterimages, when it receives a business object for update, the connector assumes that the business object represents the desired state of the data after update.

Therefore, when an integration broker sends a business object with the Update verb to the connector, the connector changes the current representation of the business object in the database so that it exactly matches the source business object. To do this, the connector changes simple attribute values and adds or removes child business objects.

For example, assume the current state of Contract 2345 in the database is as follows:



Assume further that the integration broker passes the following business object to the connector:



To process the update, the connector applies the following changes to the database:

- Update the simple attributes in the top-level Contract and Address business objects
- Create the Phone business object
- Update the simple attributes in the child business objects A, B, F and G
- Delete the child business objects C, D and E
- Create the child business objects H, I and J

Because the connector assumes that each business object it receives from the integration broker represents an afterimage, it is important to ensure that each business object sent to such a connector for updating contains valid existing child business objects. Even if none of a child business object's simple attributes have changed, the child business object must be included in the source business object.

There is a way, however, that you can prevent some connectors from deleting missing child business objects during an update operation. You can use the application-specific information for the attribute that represents the child or array of children to instruct the connector to keep child business objects that are not included in the source business object. To do so, set `KEEP_RELATIONSHIP` to true. For more information, see “Specifying an attribute’s foreign key” on page 57.

Verb processing

This section outlines the steps the connector takes when creating, retrieving, updating, or deleting a business object that it receives from an integration broker. The connector processes hierarchical business objects recursively; that is, it performs the same steps for each child business object until it has processed all individual business objects.

Note: A top-level business object that is a wrapper supports the create, retrieve, update and delete verbs. The only difference in processing a wrapper object is that the wrapper object is not processed, only the objects that it contains are processed.

Business object comparison

At various points in the processing outlined below, the connector compares two business objects to see if they are the same. For example, during an update operation, the connector determines whether a particular business object exists in an array of business objects. To perform the check, the connector compares the business object to each business object within the array. For two business objects to be identical, the following two conditions must be satisfied:

- The type of the business objects being compared must be the same. For example, a Customer business object is never considered identical to a Contact business object even if all of their attributes are exactly the same.
- All corresponding key attributes in the two business objects must contain identical values. If a key attribute is set to `CxIgnore` in both business objects, the connector considers them identical. However, if a key attribute is set to `CxIgnore` in one business object but not in the other, the business objects are not identical.

Create operations

When creating a business object, the connector returns a status either of `VALCHANGE` if the operation was successful (regardless of whether the operation caused changes to the business object), or `FAIL` if the operation failed.

The connector performs the following steps when creating a hierarchical business object:

1. Recursively inserts each single-cardinality child business object contained with ownership into the database. In other words, the connector creates the child and all child business objects that the child and its children contain.

If the business object definition specifies that an attribute represents a child business object with single cardinality and that attribute is empty, the connector ignores the attribute. However, if the business object definition requires that attribute to represent a child and it does not, the connector returns an error and stops processing.

2. Processes each single-cardinality child business object contained without ownership as follows:
 - a. Recursively attempts to retrieve the child from the database using the key values passed in by the integration broker.

- b. If the retrieve is unsuccessful, indicating that the child does not currently exist in the database, the connector returns an error and stops processing. If the retrieve is successful, the connector recursively updates the child business object.

Note: For this approach to work correctly when the child business object already exists in the application database, ensure that primary key attributes in child business objects are cross-referenced correctly on create operations. If the child business object does not already exist in the application database, set the primary key attributes to CxBlank.

3. Inserts the top-level business object in the database as follows:
 - a. Sets each of its foreign-key values to the primary-key values of the corresponding child business object represented with single cardinality. Because values in child business objects can be set by database sequences or counters or by the database itself during the creation of the child, this step ensures that the foreign-key values in the parent are correct before the connector inserts the parent in the database.
 - b. Generates a new unique ID value for each attribute that is set automatically by the database. The name of the database sequence or counter is stored in the attribute's application-specific information. If an attribute has an associated database sequence or counter, the value generated by the connector overwrites any value passed in by the integration broker. For more information on specifying a database sequence or counter, see UID=AUTO in "Application-specific information for simple attributes" on page 55.
 - c. Copies the value of an attribute to the value of another attribute as specified by the CA (CopyAttribute) parameter of the attribute's application-specific information. For more information on using the CA parameter, see CA=set_attr_name in "Application-specific information for simple attributes" on page 55.
 - d. Inserts the top-level business object into the database.

Note: A top-level business object that is a wrapper will not be inserted into the database.

4. Processes each of its single-cardinality child business objects that stores the parent/child relationship in the child, as follows:
 - a. Sets the foreign-key values in the child to reference the value in the corresponding primary-key attributes in the parent. Because the parent's primary-key values may have been generated during the creation of the parent, this ensures that the foreign-key values in each child are correct before the connector inserts the child into the database.
 - b. Inserts the child into the database.
5. Processes each of its multiple-cardinality child business objects, as follows:
 - a. Sets the foreign-key values in each child to reference the value in the corresponding primary-key attributes in the parent. Because the parent's primary-key values may have been generated during the creation of the parent, this ensures that the foreign-key values in each child are correct before the connector inserts the child into the database.
 - b. Inserts each of its multiple-cardinality child business objects into the database.

Retrieve operations

The connector performs the following steps when retrieving a hierarchical business object:

1. Removes all child business objects from the top-level business object that it received from the integration broker.
2. Retrieves the top-level business object from the database.
 - If the retrieval returns 1 row, the connector continues processing.
 - If the retrieval returns no rows, indicating that the top-level business object does not exist in the database, the connector returns `BO_DOES_NOT_EXIST`.
 - If the retrieval returns more than one row, the connector returns `FAIL`.

Note: A business object can have attributes that do not correspond to any database column, such as placeholder attributes. During retrieval, the connector does not change such attributes in the top-level business object; they remain set to the values received from the integration broker. In child business objects, the connector sets such attributes to their default values during retrieval.

Note: A top-level business object that is a wrapper must contain any attribute values from the objects at the level immediately below the wrapper object, which would be necessary to retrieve the objects, including keys and placeholder attributes. The wrapper object must have all keys and placeholder attributes populated. Simple attributes in the wrapper object that will be used as foreign keys in the objects one level below the wrapper should be marked as keys in the wrapper object.

3. Recursively retrieves all multiple-cardinality child business objects.

Note: The connector does not enforce uniqueness when populating an array of business objects. It is the database's responsibility to ensure uniqueness. If the database returns duplicate child business objects, the connector returns duplicate children.

4. Recursively retrieves each of the single-cardinality children regardless of whether the child business object is contained with or without ownership.

Note: All single cardinality child business objects are processed based on occurrence in the business object and before the parent business object is processed. Child object ownership and non-ownership do not determine the processing sequence, but do determine the type of processing.

RetrieveByContent operations

A `RetrieveByContent` verb is applicable only for the top-level business object, because the connector performs a retrieval based on attributes only in the top-level business object.

If a top-level business object uses the `RetrieveByContent` verb, all of the attributes (including non-key attributes) that are not null are used as retrieval criteria.

If more than one row is returned, the connector uses the first row as the result row and returns `MULTIPLE_HITS`.

Note: A `RetrieveByContent` verb is not applicable for a top-level business object that is a wrapper.

Update operations

When updating a business object, the connector returns a status either of VALCHANGE if the operation was successful (regardless of whether the operation caused changes to the business object), or FAIL if the operation failed. When working with an Oracle database, the connector locks data while retrieving it to ensure data integrity.

The connector performs the following steps when updating a hierarchical business object:

1. Uses the primary-key values of the source business object to retrieve the corresponding entity from the database. The retrieved business object is an accurate representation of the current state of the data in the database.

- If the retrieval fails, indicating that the top-level business object does not exist in the database, the connector returns BO_DOES_NOT_EXIST and the update fails.

Note: A top-level business object that is a wrapper does not have to exist in the database. However, it must contain any attribute values from the objects at the level immediately below the wrapper object, which would be necessary to retrieve the objects including keys and placeholder attributes. The wrapper object must have all keys and placeholder attributes populated. Simple attributes in the wrapper object that will be used as foreign keys in the objects one level below the wrapper should be marked as keys in the wrapper object.

- If the retrieval succeeds, the connector compares the retrieved business object to the source business object to determine which child business objects require changes in the database. The connector does not, however, compare values in the source business object's simple attributes to those in the retrieved business object. The connector updates the value of all non-key simple attributes.

If all the simple attributes in the top-level business object represent keys, the connector cannot generate an update query for the top-level business object. In this case, the connector logs a warning and continues to step 2.

2. Recursively updates all single-cardinality children of the top-level business object.

If the business object definition requires that an attribute represent a child business object, the child must exist in both the source business object and the retrieved business object. If it does not, the update fails, and the connector returns an error.

The connector handles single-cardinality children contained with ownership in one of the following ways:

- If the child is present in both the source and the retrieved business objects, instead of updating the already existing child in the database, the connector deletes the existing child and creates the new child.
- If the child is present in the source business object but not in the retrieved business object, the connector recursively creates it in the database.
- If the child is present in the retrieved business object but not in the source business object, the connector recursively deletes it from the database. The type of delete, physical or logical, depends on the value of its ChildUpdatePhyDelete property.

For single-cardinality children contained without ownership, the connector attempts to retrieve every child from the database that is present in the source

business object. If it successfully retrieves the child, the connector populates the child business object but does not update it, as single cardinality children contained without ownership are never modified by the connector.

3. For single-cardinality child business objects that store the relationship in the parent, sets each foreign-key value in the parent to the value of the primary key in the corresponding single-cardinality child business object. This step is necessary because single-cardinality children may have been added to the database during previous steps, resulting in the generation of new unique IDs.
4. Updates all simple attributes of the retrieved business object except those whose corresponding attribute in the source business object contain the value CxIgnore.

Because the business object being updated must be unique, the connector verifies that only one row is processed as a result. It returns an error if more than one row is returned.

5. Sets all foreign-key values in each child that stores the parent/child relationship in the child (both multiple-cardinality and single-cardinality) to the primary-key value of its corresponding parent business object. (When ICS is used as the integration broker, these values have typically been cross-referenced during data mapping.) However, this step is important to ensure that the foreign-key values of new children that store the relationship in the child are correct before the connector updates those children.
6. Processes each multiple-cardinality child of the retrieved business object in one of the following ways:
 - If the child exists in both the source and the retrieved business objects' arrays, the connector recursively updates it in the database.
 - If the child exists in the source array but not in the retrieved business object's array, the connector recursively creates it in the database.
 - If the child exists in the retrieved business object's array but not in the source array, the connector recursively deletes it from the database unless the application-specific information for the attribute that represents the child in the parent has `KEEP_RELATIONSHIP` set to true. In this case, the connector does not delete the child from the database. For more information, see "Specifying an attribute's foreign key" on page 57. The type of delete, physical or logical, depends on the value of its `ChildUpdatePhyDelete` property.

Note: The integration broker must ensure that business objects contained with multiple cardinality in the source business object are unique (that is, that an array does not contain two or more copies of the same business object). If the connector receives duplicates of a business object in a source array, it processes the business object twice, with possibly unpredictable results.

DeltaUpdate operations

DeltaUpdate verb processing is different from update verb processing as follows:

1. On a DeltaUpdate no retrieve is done before updating, as is done in update verb processing.
2. No comparisons are made between the incoming business object and the business object in the database.
3. All children will be processed based on the verb set in each child object. If a child doesn't have a verb set in it, the connector will return an error.

When delta updating a business object, the connector returns a status of either `VALCHANGE` if the operation was successful (regardless of whether the operation caused changes to the business object) or `FAIL` if the operation failed.

The connector performs the following steps when delta updating a hierarchical business object:

1. Recursively processes all single-cardinality children of the parent object. If a child is marked as `IsRequired` in the business object specification, it must be present in the inbound object. If not, the delta update will fail and the connector will return an error.
2. Sets all foreign key values in the parent that reference attributes in single-cardinality children to their corresponding child values. This is necessary because single-cardinality children may have been added to the database during the previous steps, resulting in the generation of new sequence values.
3. Updates the current object being processed via an SQL `UPDATE` statement or a stored procedure. All simple attributes of the individual business object are updated, except those attributes set to `Ignore` in the inbound business object. The connector does not compare the inbound object to the current object on an attribute level to determine which attributes need to be added to the update statement; they are all updated. Since the object being updated should be unique, the connector checks to make sure that only one row is processed as a result. An error is returned if more than one row is processed.
4. Sets all foreign key values in all cardinality N children of the current object that reference parent attributes to the corresponding parent values. Usually these values will already be cross-referenced during data mapping; however, this may not be the case for new children in cardinality N containers. This ensures that the foreign key values in all cardinality N children are correct before those children are updated.
5. Updates all cardinality N containers of the current object.

When the child objects are processed, each child's verb is taken and the appropriate operation is done. The allowed verbs on a child in `DeltaUpdate` are `Create`, `Delete` and `DeltaUpdate`.

- If a `Create` verb is found in the child, the child gets created in the database if it is an ownership child. Non-ownership children are retrieved to validate their existence in the database.
- If a `Delete` verb is found in the child, that child gets deleted.
- If a `DeltaUpdate` verb is found in the child, the child gets updated in the database.

Delete operations

When deleting a business object, the connector returns a status of `SUCCESS` if the operation was successful or `FAIL` if the operation failed. The parent business object is first retrieved and then the adapter recursively deletes all single-cardinality children that have an ownership relationship to the parent, then the parent business object itself, and finally all cardinality N children. Single-cardinality no-ownership children are never deleted. If the business object does not exist, the connector returns a `FAIL`.

The connector supports logical and physical deletes, depending on the `Status Column Name (SCN)` value in the object's application-specific information. If the `SCN` value is defined, the connector performs a logical delete. If the `SCN` value is not defined, the connector performs a physical delete.

Physical deletes: The connector performs the following steps when physically deleting a hierarchical business object:

1. Recursively deletes all single-cardinality child business objects contained with ownership.

2. Deletes the top-level business object.
3. Recursively deletes all multiple-cardinality child business objects.

Note: A top-level business object that is a wrapper does not have a corresponding database table, hence it will not be deleted from the database. Any simple attribute values for a wrapper will be ignored.

Logical deletes: When logically deleting a business object, the connector performs the following steps:

1. Issues an UPDATE that sets the business object's status attribute to the value specified by the business object's application-specific information. The connector ensures that only one database row is updated as a result, and it returns an error if this is not the case.
2. Recursively logically deletes all single-cardinality children contained with ownership and all multiple-cardinality children. The connector does not delete single-cardinality children contained without ownership.

SQL statements

The connector can use simple SQL statements for select, update, retrieve or delete operations. The column names for SQL statements are derived from an attribute's `AppSpecificInfo` property. Each query spans one table only, unless posted to a view.

Stored procedures

A stored procedure is a group of SQL statements that form a logical unit and perform a particular task. A stored procedure encapsulates a set of operations or queries for the connector to execute on an object in a database server.

The connector calls stored procedures in the following circumstances:

- Before processing a business object, to perform preparatory operational processes
- After processing a business object, to perform post-operational processes
- To perform a set of operations on a business object, instead of using a simple INSERT, RETRIEVE, UPDATE, or DELETE statement

When it processes a hierarchical business object, the connector can use a stored procedure to process the top-level business object or any of its child business objects. However, each business object or array of business objects must have its own stored procedure.

Specifying a stored procedure

This section describes the steps you must perform to cause the connector to use a stored procedure for a business object. It contains the following sections:

- "Adding attributes to the business object"
- "Syntax of a stored procedure" on page 45
- "Examples of stored procedures" on page 46
- "Specifying the stored procedure" on page 46

Adding attributes to the business object: You must add a special kind of attribute to the business object for each type of stored procedure that the connector processes. These attributes represent only the stored procedure's type and the application-specific information that defines it. These attributes do not use the application-specific information parameters available for a standard simple attribute.

Name the attribute according to the type of stored procedure to be used. For example, to cause the connector to use AfterUpdate and BeforeRetrieve stored procedures, add the AfterUpdateSP and BeforeRetrieveSP attributes.

The connector recognizes the following business object attribute names:

```
BeforeCreateSP
AfterCreateSP
CreateSP
BeforeUpdateSP
AfterUpdateSP
UpdateSP
BeforeDeleteSP
AfterDeleteSP
DeleteSP
BeforeRetrieveSP
AfterRetrieveSP
RetrieveSP
BeforeRetrieveByContentSP
AfterRetrieveByContentSP
RetrieveByContentSP
BeforeRetrieveUpdateSP
AfterRetrieveUpdateSP
RetrieveUpdateSP
```

Note: Create an attribute only for those stored procedures that you want the connector to execute. Use the application-specific information or mapping (only if ICS is used as the integration broker) to specify values for these attributes before the business object is sent to the connector. The connector must be restarted to recognize changes to these values for subsequent calls on a business object.

Syntax of a stored procedure: The syntax for specifying a stored procedure is:

```
SPN=StoredProcedureName;RS=true|false[;IP=Attribute_Name1[:Attribute_Name2[:...]]]
[;OP=Attribute_Name1|RS[:Attribute_Name2|RS[:...]]]
[;IO=Attribute_Name1[:Attribute_Name2[:...]]]
```

where:

StoredProcedureName

The name of the stored procedure.

RS

Is true if the stored procedure returns a result set or false if it does not. By default, false. If the value is true, the ColumnName property in an attribute's application-specific information points to the appropriate column in the result set. If RS is part of the output parameter list, then that particular parameter returns a result set. Only one result set OUT parameter is supported. If more than one result set is returned as an OUT parameter, only the first result set is returned and all others are ignored. Currently, this feature is supported for Oracle 8i and above, stored procedures that use the Oracle JDBC Driver. For the stored procedure in the database, the corresponding parameter should return a REF_CURSOR type.

IP

Input Parameters: The list of business object attributes whose values the connector should use as input values when executing the stored procedure.

OP	Output Parameters: The list of business object attributes to which the connector should return values after executing the stored procedure. See RS for a description of the result set.
IO	InputOutput Parameters: The list of business object attributes whose values the connector should use as input values and to which the connector should return values after executing the stored procedure.

Note: The order of StoredProcedureName, RS, and parameters is important; the order of parameters among themselves is not important. In other words, it makes no difference to the connector if the stored procedure groups all parameters of each type or intersperses the types of parameters.

When multiple parameters of the same type are grouped together, separate the values with a colon delimiter; you need not repeat the parameter's name for each value. Separate parameters of different types with a semicolon delimiter. When specifying parameter values, do not put a blank space on either side of the equal sign (=).

Examples of stored procedures: The following examples use stored procedures named CustomerInsert and VendorInsert that get values from two input attributes, and return values to four output attributes. The examples illustrate different structures for stored procedures.

- Parameters of the same type are grouped together (IP, IP, OP, OP, OP, IO):
SPN=CustomerInsert;RS=false;IP=LastName:FirstName;OP=CustomerName:CustomerID:ErrorStatus:ErrorMessage;IO=VendorID
- Parameters of the same type are interspersed (IP, OP, OP, OP, IP, IO, OP):
SPN=VendorInsert;RS=false;IP=LastName;OP=CustomerName:CustomerID:ErrorStatus;IP=FirstName;IO=VendorID;OP=ErrorMessage

The connector supports only the simple data types supported by the JDBC driver.

Specifying the stored procedure: There are two ways to specify the stored procedure name and its parameter values:

- Attribute's AppSpecificInfo property
If the length of the text that specifies the stored procedure is less than or equal to 4000 bytes, you can specify the value in the attribute's AppSpecificInfo property. You can use this property to specify the stored procedure regardless of whether the connector has polled for the business object (that is, the business object represents an application event) or has received the business object as an integration broker request.

The following example illustrates specification of the stored procedure in application-specific information. In this case, the value specified for the MaxLength property is not important to the stored procedure.

```
[Attribute]
Name = BeforeCreateSP
Type = String
MaxLength = 15
IsKey = false
IsRequired = false
AppSpecificInfo =SPN=ContactInsert;IP=LastName:FirstName;OP=CustomerName:
CustomerID:ErrorStatus:ErrorMessage

[End]
```

- Attribute's value (relevant only if ICS is used as the integration broker)
If the length of the text that specifies the stored procedure is more than 4000 bytes, you must use mapping to specify the stored procedure. You can use mapping to specify the stored procedure only if the business object represents an integration broker request. In other words, you cannot use an attribute's value to specify a stored procedure when the connector is polling for events.
If the text of the stored procedure is longer than 4000 bytes and you use mapping to specify it, remember to expand the value of the MaxLength property to accommodate the full text.

Note: If a stored procedure that handles a create, update, or delete operation is executed on a hierarchical business object containing an array of child business objects, the connector processes each child business object individually. For example, if the connector executes a BeforeCreate stored procedure, it does not process the array as a unit but processes each member in the array. When it processes a BeforeRetrieve stored procedure, the connector operates on a single business object. When it processes an AfterRetrieve stored procedure, the connector operates on all business objects returned by the retrieval.

Processing business objects using stored procedures or simple SQL statements

The following sections explain how the connector processes the stored procedures:

- "Business object create operations"
- "Business object update operations" on page 48
- "Business object delete operations" on page 48
- "Business object retrieve operations" on page 48
- "Business object RetrieveByContent operations" on page 49
- "Business object Retrieve-for-Update operations" on page 50

Business object create operations: A Create stored procedure usually returns values that the connector uses to populate the simple attributes in the top-level business object. The connector performs the following steps when processing the Create stored procedures (BeforeCreate, Create, AfterCreate):

1. Checks whether the business object contains a BeforeCreateSP attribute. If it does, calls the BeforeCreate stored procedure.
2. If the stored procedure returns values through output parameters, uses the values to set the value of simple attributes in the business object.
3. Creates the single-cardinality child business objects.
4. Sets each of the top-level business object's foreign key values to the primary-key value of each single-cardinality child business object.
5. Checks whether the business object contains a CreateSP attribute. If it does, calls the Create stored procedure to create the top-level business object. If it does not, builds and executes an INSERT statement to create the top-level business object.
6. If the Create stored procedure returns values through output parameters, uses the values to set the value of simple attributes in the business object.
7. Sets the foreign-key value in each multiple-cardinality child to the value of its parent's primary-key attribute.
8. Creates the multiple-cardinality child business objects.
9. Checks whether the business object contains an AfterCreateSP attribute. If it does, calls the AfterCreate stored procedure.

10. If the stored procedure returns values through output parameters, uses the values to set the values of simple attributes in the business object.

The connector can use values returned in step 10 to change the values of a business object that it created in steps 3 or 5.

Business object update operations: An Update stored procedure usually returns values that the connector uses to populate the simple attributes in the top-level business object. The connector performs the following steps when processing the Update stored procedures (BeforeUpdate, Update, AfterUpdate):

1. Checks whether the business object contains a BeforeUpdateSP attribute. If it does, calls the BeforeUpdate stored procedure.
2. If the BeforeUpdate stored procedure returns values through output parameters, uses the values to set the value of simple attributes in the business object.
3. Updates the single-cardinality child business objects.
4. Sets each of the top-level business object's foreign-key values to the primary-key value of each child business object contained with single cardinality.
5. Checks whether the business object contains an UpdateSP attribute. If it does, calls the Update stored procedure to update the top-level business object. If it does not, builds and executes an UPDATE statement to update the top-level business object.
6. If the Update stored procedure returns values through output parameters, uses the values to set the value of simple attributes in the business object.
7. Sets foreign-key values in the multiple-cardinality children to reference the value in the corresponding primary-key attributes in the parent.
8. Updates the multiple-cardinality child business objects.
9. Checks whether the business object contains an AfterUpdateSP attribute. If it does, calls the AfterUpdate stored procedure.
10. If the stored procedure returns values through output parameters, uses the values to set the value of simple attributes in the business object.

Business object delete operations: A Delete stored procedure does not return values to the connector. The connector performs the following steps when processing the Delete stored procedures (BeforeDelete, Delete, AfterDelete):

1. Checks whether the business object contains a BeforeDeleteSP attribute. If it does, calls the BeforeDelete stored procedure.
2. Deletes the single-cardinality child business objects.
3. Deletes the multiple-cardinality child business objects.
4. Checks whether the business object contains a DeleteSP attribute. If it does, calls the Delete stored procedure to delete the top-level business object. If it does not, builds and executes a DELETE statement.
5. Checks whether the business object contains an AfterDeleteSP attribute. If it does, calls the AfterDelete stored procedure.

Business object retrieve operations: For simple RETRIEVE operations, stored procedures can be used for top-level business object, single cardinality children, as well as multiple cardinality children. The order of the procedures is as follows:

- BeforeRetrieve
- Retrieve
- AfterRetrieve

The connector creates a temporary object to retrieve a single cardinality child business object or a multiple cardinality child business object. The connector applies the BeforeRetrieve stored procedure to the temporary business object. The AfterRetrieve stored procedure is applied to each of the child objects retrieved for the container.

The connector executes the AfterRetrieve stored procedure after it executes a Retrieve query generated dynamically from the business object meta-data or stored procedure on the business object.

According to the JDBC specification there are three types of StoredProcedure calls as follows:

- {call <spName>(?,?,?)}
- {call <spName>}
- {?= call <spName>(?,?,?)}

The connector supports the first two types. It will process the ResultSet that is returned from StoredProcedure.

In the stored procedure syntax, if RS=true, the result set from the stored procedure is processed. If RS=false, the result set is not processed. By default the value of RS is false. After the result set values are processed, the stored procedure output variables are processed. If RS=true, multiple cardinality children cannot specify the output variables in the related stored procedure.

Note: Result set processing is supported only for Retrieve verb operations and for RetrieveSP only.

Processing result set returned from retrieve stored procedure (RetrieveSP)::

ResultSetMetaData is obtained for the result set returned from the stored procedure. Values of all the columns in the result set are obtained and set on the corresponding attribute of the business object. The ColumnName property of an attribute's application-specific information should contain the ResultSet column name to match the attribute to the column.

For single cardinality objects, the corresponding result set should consist of only one row. If multiple rows are returned in the result set, an error is reported.

For multiple cardinality children, multiple rows can be returned through the result set. For each row returned, a new object is created and added to the container. The container is then added to the parent object at the required attribute index.

Business object RetrieveByContent operations: For simple RetrieveByContent operations, stored procedures can be used only for the top-level business object and its single-cardinality children; that is, they cannot be used to return a result set or multiple rows. The order of the procedures is as follows:

- BeforeRetrieveByContent
- RetrieveByContent
- AfterRetrieveByContent

The connector creates a temporary object to retrieve a single cardinality child business object or a multiple cardinality child business object. For multiple cardinality business objects, the connector applies the BeforeRetrieveByContent stored procedure to the temporary business object. The AfterRetrieveByContent stored procedure is applied to each of the child objects retrieved for the container.

The connector executes the `AfterRetrieveByContent` stored procedure after it executes a `RetrieveByContent` query generated dynamically from the business object meta-data or stored procedure on the business object. In this case, even though the retrieval of a hierarchical business object also retrieves its child business objects, the connector executes the `AfterRetrieveByContent` stored procedure on every business object present in the array.

Business object Retrieve-for-Update operations: The following stored procedures are called on the top-level business object and retrieve all child business objects in the same way as the simple `Retrieve`.

The order of the procedures is as follows:

- `BeforeRetrieveUpdate`
- `RetrieveUpdate`
- `AfterRetrieveUpdate`

These stored procedures perform the same operations as `BeforeRetrieve` and `AfterRetrieve`. They have distinguishing names so that you can create separate attributes to cause the connector to perform both `BeforeRetrieve` and `BeforeRetrieveUpdate` operations, as well as `AfterRetrieve` and `AfterRetrieveUpdate` operations.

The connector creates a temporary object to retrieve a single cardinality child business object or a multiple cardinality child business object. For multiple cardinality business objects, the connector applies the `BeforeRetrieveUpdate` stored procedure to the temporary business object. The `AfterRetrieveUpdate` stored procedure is applied to each of the child objects retrieved for the container.

The connector executes the `AfterRetrieveUpdate` stored procedure after it executes a `RETRIEVE` query generated dynamically from the business object meta-data or stored procedure on the business object. In this case, even though the retrieval of a hierarchical business object also retrieves its child business objects, the connector executes the `AfterRetrieveUpdate` stored procedure on every business object present in the array.

Transaction commit and rollback

Whenever the connector receives a business object for processing, it begins a transaction block. All SQL statements that the connector executes while processing that business object are encapsulated within the transaction block. When the connector finishes processing the business object, it commits the transaction block if the processing was successful, or rolls back the transaction if it encountered an error.

Business object attribute properties

Business object architecture defines various properties that apply to attributes. This section describes how the connector interprets several of these properties and describes how to set them when modifying a business object.

Name property

Each business object attribute must have a unique name.

Type property

Each business object attribute must have a type, such as Integer, String, or the type of a child business object. When the connector encounters an attribute of type Date, Long Text, or String, the connector wraps the value in quotation marks and handles the value as character data.

Cardinality property

Each business object attribute that represents a child or array of child business objects has the value of 1 or n, respectively, in this attribute. All attributes that represent child business objects also have a ContainedObjectVersion property (which specifies the child's version number) and a Relationship property (which specifies the value Containment).

Max length property

If the attribute is of type String, this property specifies the maximum length allowed for the attribute's value.

Key property

At least one simple attribute in each business object must be specified as the key. To define an attribute as a key, set this property to Yes. If the business object attribute is of type String, it is recommended that the data type in the database is of type Varchar instead of char.

Note: The connector does not support specifying an attribute that represents a child business object or an array of child business objects as a key attribute.

If the key property is set to true for a simple attribute, the connector adds that attribute to the WHERE clause of SELECT, UPDATE, RETRIEVE, and DELETE SQL statements that it generates while processing the business object.

If the key property is set to true for an attribute in a child that stores the parent/child relationship in the child (both multiple-cardinality and single-cardinality), the connector uses the parent's primary keys in the WHERE clause of the SELECT statement, and it does not use the Key property. For information on specifying the name of business object attributes whose values are used to set the child's foreign-key attributes, see "Application-specific information at the attribute level" on page 55.

Foreign key property

The connector uses this property to determine whether an attribute is a foreign key.

Required property

The Required property specifies whether an attribute must contain a value.

If this property is specified for an attribute that represents a single-cardinality child business object, the connector requires the parent business object to contain a child business object for this attribute.

When the connector receives a business object with a Create request, the connector causes the Create operation to fail if both of the following conditions are true:

- The business object does not have a valid value or a default value for a required attribute.

- Application-specific information does not specify that the connector generate the unique ID.

When the connector receives a business object with a Retrieve request and the business object does not have a valid value or a default value for a required attribute, the connector causes the retrieval operation to fail.

The connector does not use this property for attributes that contain an array of child business objects.

Note: If the key attribute uses a sequence or counter or is populated by the database (UID=AUTO), it should not be marked as Required.

AppSpecificInfo

For information on this property, see “Application-specific information at the attribute level” on page 55.

Default value property

This property specifies a default value that the connector uses to populate a simple attribute if it is not populated with a value from the database table. The connector does not evaluate this property for attributes that represent a child business object or an array of child business objects.

The connector evaluates this property only if the UseDefaults configuration property is set to true. For more information, see Table 6 on page 15.

Special attribute value

Simple attributes in business objects can have the special value, CxIgnore. When it receives a business object from the integration broker, the connector ignores all attributes with a value of CxIgnore. It is as if those attributes were invisible to the connector.

When the connector retrieves data from the database and the SELECT statement returns a null value for an attribute, the connector sets the value of that attribute to CxIgnore by default. If a value has been specified for the UNVL parameter of the attribute’s application-specific information, the connector uses that value to represent the null.

Because the connector requires every business object to have at least one primary-key attribute, developers should ensure that WebSphere Business Integration Adapter business objects passed to the connector have at least one primary key that is not set to CxIgnore. The only exception to this requirement is a business object whose primary key is to be generated by the connector using a counter or sequence, or is generated by the database.

When the connector inserts data into the database and a business object attribute has no value specified, it uses the value specified by the attribute’s UseNullValue property. For more information about UseNullValue, see UNVL=value in Table 11 on page 55.

Business object application-specific information

Application-specific information in business object definitions provides the connector with application-dependent instructions on how to process business objects. The connector parses the application-specific information from the attributes or verb of a business object or from the business object itself to generate queries for create, update, retrieve, and delete operations.

The connector stores some of the business object's application-specific information in cache and uses this information to build queries for all the verbs.

If you extend or modify an application-specific business object, you must make sure that the application-specific information in the business object definition matches the syntax that the connector expects.

This section provides information on the object-level, attribute, and verb application-specific information format for business objects supported by the connector.

Table 10 provides an overview of the functionality available in business object application-specific information.

Table 10. Overview of application-specific information in supported business objects

Scope of application-specific information	Functionality
Entire business object	Specifies: <ul style="list-style-type: none">• The name of the corresponding database table.• Defines the column whose value the connector uses in the WHERE clause to perform a logical (or soft) delete.• That the top-level business object is a wrapper.
Simple attributes	Specifies: <ul style="list-style-type: none">• The database column name for an attribute.• The foreign key relationship between an attribute in the current business object and a parent or child business object.• Automatic generation of unique identifier values.• The name of another attribute within the same business object whose value the connector must use to set the value of the current attribute.• Whether to use the current attribute when sorting a retrieval.• The value to use when the value of the current attribute is null.• String substitution behavior.• Whether to use the LIKE operator or = operator when comparing strings.• The value to use as the wildcard position when the LIKE operator is used.
Attributes that contain a child or an array of child business objects	Specifies whether a single-cardinality child is owned by the parent. Specifies whether the connector deletes child data during an update operation if the data is not represented in the source business object.

Table 10. Overview of application-specific information in supported business objects (continued)

Scope of application-specific information	Functionality
Business object verb	Used only for the Retrieve verb, this text specifies the attributes to be included in the WHERE clause for a retrieval. You can also specify operators and attribute values.

The following sections discuss this functionality in more detail.

Application-specific information at the business-object level

Application-specific information at the business-object level allows you to:

- Specify the name of the corresponding database table.
- Provide the information necessary to perform a physical or logical delete.
- Specify that the top-level business object is a wrapper object.

At the business-object level, application-specific information format consists of parameters separated by colon (:) or semicolon (;) delimiters:

```
TN=TableName; SCN=StatusColumnName:StatusValue
```

where *TableName* identifies the database table, *StatusColumnName* is the name of the database column used to perform logical deletes, and *StatusValue* is the value that signifies that a business object is inactive or deleted.

For example, assume that a Customer business object has the following value specified for its business object application-specific information:

```
TN=CUSTOMER; SCN=CUSTSTATUS:DELETED
```

Assume also that the connector receives a request to delete the customer. Such a value causes the connector to issue the following SQL statement:

```
UPDATE CUSTOMER SET CUSTSTATUS = 'DELETED' WHERE CUSTOMER_ID = 2345
```

If the SCN parameter is not included or no value is specified for it, the connector physically deletes the business object from the database. In other words, if the business object with the Delete verb includes the SCN parameter in its application-specific information, the connector performs a logical delete. If the business object with the Delete verb does not include the SCN parameter in its application-specific information, the connector performs a physical delete.

Both update and delete operations may use the value of the SCN property:

- When performing an update, the connector uses the value of its `ChildUpdatePhyDelete` property to determine whether to physically or logically delete missing child data. If logically deleting the child data, it uses the value of its SCN parameter to obtain the name of the status column and the text of the status value. For more information, see “Update operations” on page 41.
- When performing a delete, the connector uses the value of its SCN parameter to determine whether to physically or logically delete the entire business object. If the SCN parameter contains a value, the connector performs a logical delete. If the SCN parameter does not contain a value, the connector performs a physical delete. For more information, see “Delete operations” on page 43.

At the business-object level, application-specific information may be used to specify a wrapper:

```
WRAPPER=true|false
```

If the wrapper parameter is set to true, the top-level business object is a wrapper object. The wrapper object is not represented by a database table or view. A wrapper is used as a container for unrelated business objects. The connector ignores the top-level object and processes only the children. The wrapper object may contain N cardinality or N-1 cardinality entities or both.

Application-specific information at the attribute level

The application-specific information for attributes differs depending on whether the attribute is a simple attribute or an attribute that represents a child or an array of child business objects. The application-specific information for an attribute that represents a child also differs depending on whether the parent/child relationship is stored in the child or in the parent. For information on application-specific information for attributes that represent a child or array of child business objects, see “Specifying an attribute’s foreign key” on page 57.

Application-specific information for simple attributes

For simple attributes, application-specific information format consists of eleven name-value parameters, each of which includes the parameter name and its value. Each parameter set is separated from the next by a colon (:) delimiter.

The format of attribute application-specific information is shown below. Square brackets ([]) surround an optional parameter. A vertical bar (|) separates the members of a set of options. Reserve the colon as a delimiter.

```
CN=col_name:[FK=[fk_object_name.]fk_attribute_name]:
[UID=[AUTO|uid_name] schema_name.uid_name[=UseIfMissing]|CW.uidcolumnname
[=UseIfMissing]]]:
[PH=true|false]:[CA=set_attr_name] ..set_attr_name]:[OB=[ASC|DESC]]:[UNVL=value]:
[ESC=true|false]:[FIXEDCHAR=true|false]:
[BYTEARRAY=true|false]:[USE_LIKE=true|false]:
[WILDCARD_POSITION=non-negative number|NONE|BEGIN|END|BOTH]]:
[CLOB=true]
```

The only required parameter for a simple attribute that you want the connector to process is the column name. For example, to specify only the column name, use the following format:

```
CN=customer_id
```

Table 11 describes each name-value parameter.

Table 11. Name-value parameters in attribute application-specific information

Parameter	Description
CN=col_name	The name of the database column for this attribute.
FK=[fk_object_name.]fk_attribute_name	The value of this property depends on whether the parent/child relationship is stored in the parent business object or the child. If an attribute is not a foreign key, do not include this parameter in the application-specific information. For more information, see “Specifying an attribute’s foreign key” on page 57.

Table 11. Name-value parameters in attribute application-specific information (continued)

Parameter	Description
UID=AUTO	The connector uses this parameter to generate the unique ID for the business object. If an attribute does not require generation of a unique ID, do not include this parameter in the application-specific information. See the PreserveUIDSeq property description for details on preserving the unique ID during business object processing. For more information, see “Generating a business object’s unique identifier” on page 60. Note: CW is a keyword used to represent the type of UID and does not represent the tablename.
UID= <i>uid_name</i> <i>schema_name.uid_name</i> [=UseIfMissing]	
UID=CW. <i>uidcolumnname</i> [=UseIfMissing]	
PH=true false	If PH=true, then the corresponding simple attribute is a placeholder attribute. A simple attribute is also a placeholder if its ASI is blank or null.
CA= <i>set_attr_name</i> .. <i>set_attr_name</i>	If <i>set_attr_name</i> is set to the name of another attribute within the current individual business object, the connector uses the value of the specified attribute to set the value of this attribute before it adds the business object to the database during a Create operation. The value of <i>set_attr_name</i> cannot reference an attribute in a child business object, but it can reference an attribute in the parent business object if there if <i>set_attr_name</i> is preceded by the two periods. If you do not include this parameter in the application-specific information, the connector uses the value of the current attribute without copying the attribute’s value (CA) from another attribute.
OB=[ASC DESC]	If a value is specified for this parameter and the attribute is in a child business object, the connector uses the value of the attribute in the ORDER BY clause of retrieval queries. The connector can retrieve child business objects in ascending order or descending order. Use ASC to specify retrieval in ascending order. Use DESC to specify retrieval in descending order. If you do not include this parameter in the application-specific information, the connector does not use this attribute when specifying retrieval order.
UNVL= <i>value</i>	Specifies the value the connector uses to represent a null when it retrieves a business object with null-valued attributes. If you do not include this parameter in the application-specific information, the connector inserts a CxIgnore for the attribute’s value.
ESC=[true false]	Determines whether the connector replaces all instances of each character identified in the ReplaceAllStr property with the substitution strings also specified in the ReplaceStrList property. If this parameter does not contain a value, the connector uses the value of the ReplaceStrList property to make this determination. Note: The ESC parameter and the ReplaceAllStr and ReplaceStrList properties provide support for database escape character functionality (for example, escaping single quotes). Because the same functionality is also available from the Prepared Statements provided by the JDBC driver, these properties will be deprecated in future releases of the connector. The connector currently supports the use of the JDBC Prepared Statements.
FIXEDCHAR=true false	Specifies whether the attribute is of fixed length when the columns in the table are of type CHAR, not VARCHAR. For example, if a particular attribute is linked to a column that is of type CHAR, the connector expects FIXEDCHAR in length; for the application specific information of that attribute specify FIXEDCHAR=true. Ensure that the MaxLength property of the attribute is of the CHAR length, which is specified in the database. By default, FIXEDCHAR=false.
BYTEARRAY=true false	If BYTEARRAY=true, the connector will read and write binary data to the database and will send that data as a string to ICS or WebSphere MQ Integrator Broker. BYTEARRAY=false is the default. For more information, see “Working with binary data” on page 61.

Table 11. Name-value parameters in attribute application-specific information (continued)

Parameter	Description
USE_LIKE=true false	Specifies whether the connector compares strings using the = operator or the LIKE operator. If USE_LIKE is set to true, wildcard queries can be performed by setting WILDCARD_POSITION. If USE_LIKE is set to false, the =operator will be used.
WILDCARD_POSITION=non-negative number NONE BEGIN END BOTH	If USE_LIKE is true, the WILDCARD_POSITION is used to specify the position that is the wildcard. This value can be any non-negative number, NONE, BEGIN, END, or BOTH. For example, using BEGIN will place the wildcard character in the first position of the string (%string). Using END will place the wildcard character in the last position of the string (string%). Using BOTH will place wildcard characters in both the first and last position in the string (%string%).
CLOB=true	<p>Only applicable for String Attribute Type. Specifies that the database column that corresponds to this attribute is a CLOB datatype.</p> <p>Note: A CLOB datatype is defined as follows:</p> <ul style="list-style-type: none"> • The CLOB attribute has a String Type whose length is used to define the length of the CLOB • The CLOB attribute has ASI=CN=xyz; CLOB=true • Any other attribute type with reference to CLOB in the ASI would result in an error • CLOB=false would result in an error <p>A regular String Type would be the same and with no reference to CLOB in the ASI. CLOB datatypes of 4k and larger can be inserted or updated. But they can be used only with Oracle and require the latest thin driver withCLOB support. Using any other driver may cause errors.</p>

Note: If none of the application-specific information in any of a business object's attributes cause the connector to build or execute a query, the connector logs a warning and continues operating. It does not throw an exception or return a failure.

Specifying an attribute's foreign key: The value of this property depends on whether the parent/child relationship is stored in the parent business object or the child:

- Stored in the parent—set the value to include both the type of the child business object and the name of the attribute in the child to be used as the foreign key.
- In the child—set the value to include only the name of the attribute in the parent to be used as the foreign key.

If the value of `fk_object_name` does not match the type of the child business object, and the value of `fk_attribute_name` does not match the name of the attribute in the parent or child (as applicable), the connector cannot process this attribute as a foreign key. The case of the business object's name and the attribute's name is significant.

For example, assume that the Customer business object contains the `Addr[1]` attribute, which represents the Address child business object, and the `AID` attribute, which stores the primary key of the child business object as a foreign key. In this case, the application-specific information of the parent's foreign key attribute must contain the type of the child business object (Address) as well as the name of its primary key attribute (ID). In this example, the application-specific information of the `AID` attribute would include `FK=Address.ID`.

Naming a foreign key attribute: Multiple parent business objects can contain the same child business object, regardless of whether the child is stored with single cardinality or multiple cardinality, and regardless of whether the parent/child relationship is stored on the parent or on the child. However, all parent business objects that store the parent/child relationship must use identically named attributes to contain the child's primary key. Moreover, all child business objects that store the parent/child relationship must use identically named attributes to contain the parent's primary key. Figure 5 illustrates these relationships.

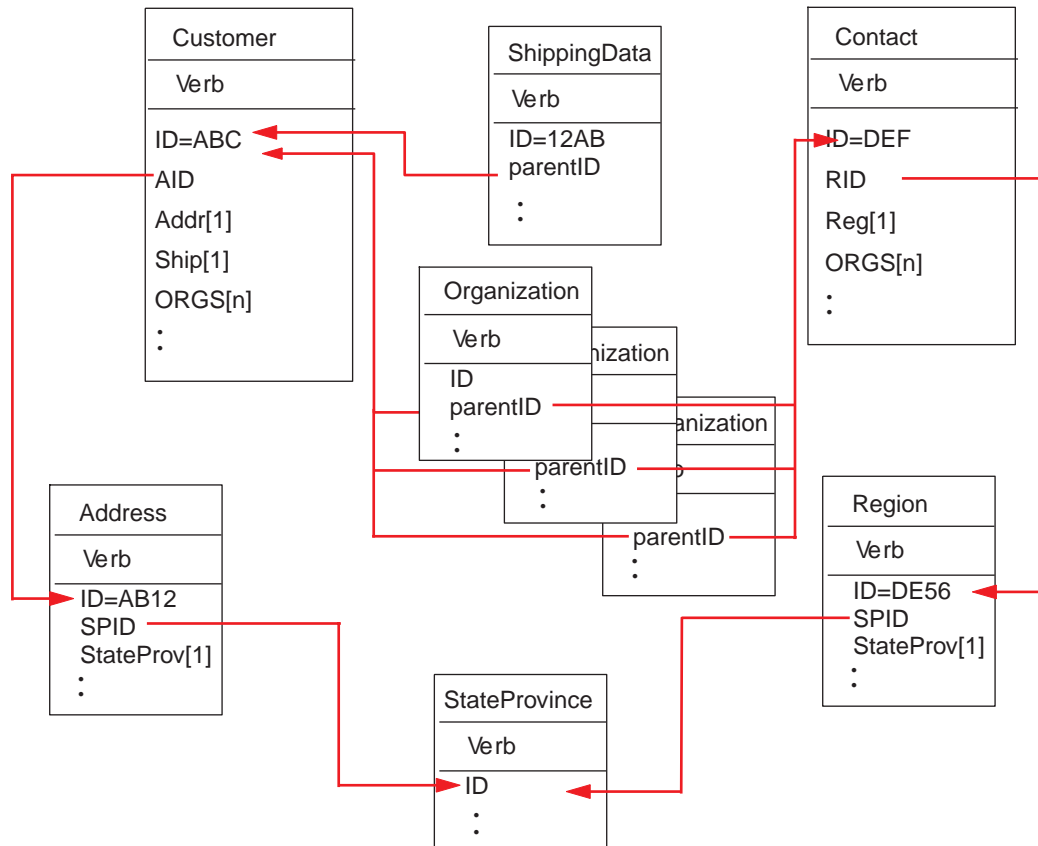


Figure 5. Example of relationships among business objects

Figure 5 illustrates the following relationships:

- The ORGS[n] attribute of Customer ABC and Contact DEF represents an array of Organization business objects. The foreign key value for each business object in the array of Organizations corresponds to the primary key value in the ID attribute in the Customer and Contact business objects. In this case, each business object in the array is contained by multiple parents.

The application-specific information for the ORGS attribute might be:

KEEP_RELATIONSHIP=true

For more information on the KEEP_RELATIONSHIP parameter, see “Application-specific information for attributes that represent children” on page 61.

The application-specific information for the parentID attribute of each child in the array of Organizations contains the name of the column in the database that corresponds to the current attribute, and specifies the current attribute's foreign key by containing the name of the parent's primary key attribute; for example:

CN=ORG_ID:FK=ID

Note: For multiple business objects to contain the same child (where the parent/child relationship is stored in the child), all parent business objects must use an identically named attribute to contain the foreign key for the child. The foreign key parameter of that child's application-specific information identifies only the attribute's name and not the type of the parent business object. The connector assumes that the direct parent is the owner of each child.

- The Addr[1] attribute of Customer represents the Address business object with ownership. The AID attribute of Customer identifies the primary key of the Address business object as a foreign key in the parent. In this case, the parent's foreign key attribute must contain the type of the child business object as well as the name of its primary key attribute. The single-cardinality child, Address, is contained by only one parent.

The application-specific information for the Addr attribute is:

CONTAINMENT=OWNERSHIP

The application-specific information for the AID attribute contains the name of the column in the database that corresponds to the current attribute, and specifies the current attribute's foreign key by containing the type of the child business object and the name of its primary key attribute; for example:

CN=FK_AD:FK=Address.ID

The application-specific information for the child's primary-key attribute is
CN=pk

- The StateProv[1] attributes of the Address and Region business objects represent the StateProvince business object without ownership. The SPID attributes of the Address and Region business objects contain the type of the child business object (StateProvince) and the name of its primary key attribute, which serve as the parent's foreign key. The same single-cardinality child, StateProvince, is contained by multiple parents.

The application-specific information for the SPID attribute is:

CONTAINMENT=NO_OWNERSHIP

For more information on the CONTAINMENT parameter, see "Application-specific information for attributes that represent children" on page 61.

The application-specific information for the Address SPID attribute contains the name of the column in the database that corresponds to the current attribute, and specifies the current attribute's foreign key by containing the type of the child business object and the name of its primary key attribute; for example:

CN=FK_SP:FK=StateProvince.ID

The application-specific information for the child's primary key attribute is:
CN=SP_ID

Note: For multiple business objects (that store the parent/child relationship in the parent) to contain the same child, all child business objects must use an identically named attribute to contain the foreign key for the parent.

- The Ship[1] attribute of Customer represents a ShippingData business object that contains the customer's shipping information. The ID attribute of Customer functions as the foreign key for the shipping data. In this case, because

ShippingData cannot exist independently of its parent and is created only after its parent is created, the parent/child relationship is stored in the child.

The application-specific information for the child's parentID attribute contains the name of the column in the database that corresponds to the current attribute, and specifies the current attribute's foreign key by containing the name of its parent's primary key attribute; for example:

```
CN=SD_ID:FK=ID
```

Generating a business object's unique identifier: The connector uses the UID parameter to generate the unique ID for the business object. The connector generates unique IDs by using sequences (as Oracle does), or counters (which are structured as tables), and then issues the INSERT statement.

IBM DB2 and Microsoft SQL Server do not require that the ID be passed in an INSERT statement. Instead, they generate the ID at the time of creation. After successful creation of the business object, the connector can retrieve and use this value.

The connector uses a sequence or counter to generate the ID value and then issues the INSERT statement:

- If UID = AUTO, the database generates the ID and the connector must retrieve it. This setting is only available for IBM DB2 and Microsoft SQL Server databases.
- If UID = *uid_name*, the value of *uid_name* provides the name of the Oracle sequence that the connector uses to generate a unique ID for the attribute. After the connector fetches the sequence value, it populates the key attribute and issues the INSERT statement. This syntax is currently used only for Oracle databases.
- If UID = *uid_name=UseIfMissing* and if the value of the attribute is not CxIgnore, the connector uses the attribute's value rather than generating a unique ID. The =UseIfMissing parameter cannot contain blanks and is case-insensitive. This option is available only for Oracle databases.
- If UID=CW.*uidcolumnname*, the connector uses a counter table to generate a unique ID for the attribute. The table, whose name is configurable, is created with a single column named id. You can customize the table to add a column for each attribute that requires generation of a UID. Use the *uidcolumnname* parameter to specify the name of the column for the connector to use when generating the unique ID. Note that the connector supports only the numeric data type for columns that require generation of a UID.

For information on configuring the table's name, see UniqueIDTableName. The scripts for installing this table are:

```
\connectors\JDBC\dependencies\uid_table_oracle.sql
```

```
\connectors\JDBC\dependencies\uid_table_mssqlserver.sql
```

```
\connectors\JDBC\dependencies\uid_table_db2.sql
```

- If UID=CW.*uidcolumnname=UseIfMissing* and if the value of the attribute is not CxIgnore, the connector uses the attribute's value rather than generating a unique ID. The =UseIfMissing parameter cannot contain blanks and is case-insensitive.

See the "PreserveUIDSeq" on page 22 property for information on preserving the unique ID sequence during processing.

Application-specific information for attributes that represent children

Attributes that represent a single-cardinality child business object can specify whether the child is owned by the parent or shared among multiple parents.

Attributes that represent a single-cardinality child or an array of child business objects can specify the connector's behavior when updating the parent and a subset of the children.

Attributes that represent a single-cardinality child business object: The format of the application-specific information for attributes that represent a single-cardinality child business object is:

CONTAINMENT= [OWNERSHIP|NO_OWNERSHIP]

Set CONTAINMENT to OWNERSHIP to represent a single-cardinality relationship where the parent owns the child business object. Set CONTAINMENT to NO_OWNERSHIP to represent a single-cardinality relationship where the parent shares the child business object. Do not include the CONTAINMENT parameter when you represent a single-cardinality relationship that stores the relationship in the child rather than in the parent.

For more information, see "Single-cardinality relationships and data without ownership" on page 33 and "Single-cardinality relationships that store the relationship in the child" on page 35.

Attributes that represent a child that stores the parent's key: For Update operations on an array of business objects that store the parent/child relationship in the child, there is a special value for the attribute that represents the child: you can set KEEP_RELATIONSHIP to true to prevent the connector from deleting existing child data that is not represented in the source business object.

For example, assume an existing contract is associated with an existing site, such as New York. Assume further that the connector receives a request to update a Contract business object that contains a single child business object that associates San Francisco as the site. If KEEP_RELATIONSHIP evaluates to true for the attribute that represents the site data, the connector updates the contract to add its association with San Francisco and does not delete its association with New York.

However, if KEEP_RELATIONSHIP evaluates to false, the connector deletes all existing child data that is not contained in the source business object. In such a case, the contract is associated only with San Francisco.

The format for this application-specific information is:

KEEP_RELATIONSHIP=[true|false]

Case is ignored in checking for this application-specific information.

Working with binary data: If BYTEARRAY=true, the connector will read and write binary data to the database. Since there is no support for binary data in the current version of the WebSphere business integration system, the binary data is converted to a String and then sent to the integration broker. The format of this string is a hexadecimal number with 2 characters per byte. For example, if the binary data in the database is 3 bytes with the (decimal) values (1, 65, 255), the string will be "0141ff".

Application-specific information format for verbs

The connector uses verb application-specific information for the Retrieve and RetrieveByContent verbs. This text allows you to specify the attributes to be included in the WHERE clause for a retrieval. You can also specify operators and attribute values.

The syntax for application-specific information for the Retrieve and RetrieveByContent verbs is shown below:

```
[condition_variable conditional_operator @ [...]:[..]attribute_name [, ...]]
```

where:

<i>condition_variable</i>	The name of the database column.
<i>conditonal_operator</i>	The operator supported by the database, for example =, >, OR, AND, and IN (<i>value1</i> , <i>value2</i>).
@	A variable that is substituted with the value retrieved by <code>getAttrValue(<i>attribute_name</i>)</code> . The substitution is positional; that is, the connector substitutes the first @ with the value of the first <i>attribute_name</i> variable specified after the : delimiter.
..	The attribute specified in the <i>attribute_name</i> variable belongs to the immediate parent business object; if this value is missing, the attribute belongs in the current business object.
<i>attribute_name</i>	The name of the attribute whose value the connector substitutes for @.

To understand the syntax of this property, assume that an Item business object has an `item_id` attribute whose value is XY45 and a `Color` attribute whose value is RED. Assume further that you specify the Retrieve verb's `AppSpecificInfo` property as:

```
Color='RED'
```

The above application-specific information value causes the connector to build the following WHERE clause for a retrieval:

```
where item_id=XY45 and Color = 'RED'
```

For a more complicated example, assume that the Customer business object has a `customer_id` attribute whose value is 1234 and a `creation_date` attribute whose value is 01/01/90. Assume also that this business object's parent has a `quantity` attribute whose value is 20.

Assume further that you specify the Retrieve verb's `AppSpecificInfo` property as:

```
creation_date > @ OR quantity = @ AND customer_status IN ('GOLD', 'PLATINUM') : creation_date, ..quantity
```

The above application-specific information value causes the connector to build the following WHERE clause for a retrieval:

```
where customer_id=1234 and creation_date > '01/01/90'  
OR quantity = 20 AND customer_status IN ('GOLD', 'PLATINUM')
```

The connector gets the date value ('01/01/90') from the `creation_date` attribute in the current business object. It gets the quantity value (20) from the `quantity` attribute in the parent business object (as indicated by `..quantity` in the application-specific information).

After the connector parses the application-specific information for the `Retrieve` verb, it adds the text to the `WHERE` clause of the `RETRIEVE` statement that it constructs from the business object's primary or foreign keys. The connector adds the leading `AND` to the `WHERE` clause. The value of the application-specific information must be valid SQL syntax. In the case of `RetrieveByContent`, the application-specific information is added to the `WHERE` clause of the `RETRIEVE` statement that it constructs from the business object's attributes that have their values populated.

The `WHERE` clause can also refer to placeholder attributes instead of the actual attributes in the parent business object. These placeholders do not have any application-specific information. An attribute can be a placeholder if it satisfies one of the following conditions for its ASI:

1. Simple attribute with `ASI=null` or ''
2. Simple attribute with `ASI=PH=TRUE`

For example: An `Order` business object contains a multiple cardinality line item business object, and retrieval of only specific line items is needed. This retrieval can be handled through a placeholder attribute in the `Order` business object. This placeholder is required in the parent object because the child objects are all pruned. The placeholder attribute can be populated at runtime by the integration broker with a list of the specific line items, separated by a comma (,).

For this example, you would add the following information to the `WHERE` clause for the retrieve verb on the child line item business object:

```
line_item_id in(@,@, @):..placeholder1,..placeholder2,..placeholder3
```

Where `line_item_id in` is the ID in the child business object, `placeholder` is the attribute in the parent. If `placeholder` contains the values 12,13,14 the query would select the following from the `WHERE` clause:

```
line_item_id in(12,13,14)
```

Where `SELECT:..FROM:..WHERE x in (1,2,3)` is a standard database SQL syntax.

In the `RetrieveByContent` verb, if the length of the `WHERE` clause is 0, the connector will use the application-specific information in the `WHERE` clause of the `RETRIEVE` statement. With this feature, the user can send a business object with no attribute values populated and specify verb application-specific information for `RetrieveByContent`, and the connector will build the `WHERE` clause based on what was specified in the verb application-specific information alone.

Chapter 4. Generating business object definitions using JDBCODA

This chapter describes JDBCODA, an object discovery agent (ODA), which generates business object definitions for the connector for JDBC. Because the connector works with objects that are table-based or view-based, JDBCODA uses database tables and views to discover business object requirements specific to its JDBC data source.

Note: Familiarity with database concepts and JDBC drivers (for configuration purposes) can aid in understanding how JDBCODA operates.

This chapter contains the following sections:

- “Installation and usage”
- “Using JDBCODA in business object designer” on page 68
- “Contents of the generated definition” on page 74
- “Sample business object definition file” on page 77
- “Inserting attributes that contain child business objects” on page 78
- “Adding information to the business object definition” on page 78

Installation and usage

This section discusses the following:

- “Installing JDBCODA”
- “Before using JDBCODA” on page 66
- “Launching JDBCODA” on page 67
- “Running multiple instances of JDBCODA” on page 67
- “Working with error and trace message files” on page 67

Installing JDBCODA

To install JDBCODA, use the Installer for IBM WebSphere Business Integration Adapter. Follow the instructions in the *System Installation Guide for UNIX or for Windows*. When the installation is complete, the following files are installed in the directory on your system where you have installed the product:

- ODA\JDBC\JDBCODA.jar
- ODA\messages\JDBCODAAgent.txt
- ODA\messages\JDBCODAAgent_11_11.txt (message files specific to a language (*ll*) and a country or territory (*TT*)).
- ODA\JDBC\start_JDBCODA.bat (Windows only)
- ODA\JDBC\start_JDBCODA.sh (UNIX only)
- bin\CWODAEV.bat (Windows only)
- bin\CWODAEV.sh (UNIX only)

Note: Except as otherwise noted, this document uses backslashes (\) 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.

Before using JDBCODA

Before you can run JDBCODA, you must:

- Install the appropriate JDBC driver.

Important: JDBCODA can connect to any database using a JDBC driver that supports JDBC 2.0 or above.

- Because JDBCODA generates business object names and attribute names from the names of corresponding database tables and columns, and because business object names and attribute names must be in ISO Latin-1, verify that the appropriate database components have Latin-1 names. If they do not, you have the following choices:
 - Create the business object definition manually in Business Object Designer.
 - Edit the definition generated by JDBCODA so that all business object names and attribute names are in Latin-1.
- Open for editing the shell or batch file and configure the values described in Table 12.

Table 12. Shell and batch file configuration variables

Variable	Explanation	Example
AGENTNAME	Name of the ODA	UNIX: AGENTNAME=JDBCODA Windows: set AGENTNAME=JDBCODA
AGENT	Name of the ODA's jar file	UNIX: AGENT=\$CROSSWORLDS/ODA/JDBC/JDBCODA.jar Windows: set AGENT= %CROSSWORLDS%\ODA\JDBC\JDBCODA.jar
DRIVERPATH	Path of JDBC driver library; JDBCODA uses the driver classes to establish a connection to a specified database	UNIX: DRIVERPATH=\$CROSSWORLDS/lib/ \xwutil.jar:\$CROSSWORLDS/lib/ \xwbase.jar:\$CROSSWORLDS/lib/ \xwsqserver.jar:\$CROSSWORLDS/lib/ \spy/lib/spy.jar Windows: set DRIVERPATH=%CROSSWORLDS% / lib\xwutil.jar;%CROSSWORLDS%\lib\ / xwbase.jar;%CROSSWORLDS%\lib\ / xwsqserver.jar;%CROSSWORLDS%\lib\ / spy\lib\spy.jar
DRIVERLIB	Path of the native libraries used by the JDBC driver	UNIX: DRIVERLIB=\$CROSSWORLDS/bin/db2jdbc.dll Windows: DRIVERLIB=%CROSSWORLDS%\bin\db2jdbc.dll

After installing the JDBC driver and setting configuration values in the shell or batch file, you must do the following to generate business objects:

1. Launch the ODA.
2. Launch Business Object Designer.
3. Follow a six-step process in Business Object Designer to configure and run the ODA.

The following sections describe these steps in detail.

Launching JDBCODA

You can launch the JDBCODA with the startup script appropriate for your operating system.

UNIX:

```
start_JDBCODA.sh
```

Windows:

```
start_JDBCODA.bat
```

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

Running multiple instances of JDBCODA

It is recommended that you change the name of the ODA when you run multiple instances of it. To create additional uniquely named instances of JDBCODA:

- Create a separate script or batch file for each instance.
- Specify a unique name in the AGENTNAME variable of each script or batch file.

It is recommended that you prefix each name with the name of the host machine when you run ODA instances on different machines.

Figure 6 on page 69 illustrates the window in Business Object Designer from which you select the ODA to run.

Working with error and trace message files

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

```
AgentNameAgent.txt
```

If you create multiple instances of the ODA script or batch file and provide a unique name for each represented ODA, you can have a message file for each ODA instance. Alternatively, you can have differently named ODAs use the same message file. There are two ways to specify a valid message file:

- If you change the name of an ODA and do not create a message file for it, you must change the name of the message file in Business Object Designer as part of ODA configuration. Business Object Designer provides a name for the message file but does not actually create the file. If the file displayed as part of ODA configuration does not exist, change the value to point to an existing file.
- You can copy the existing message file for a specific ODA, and modify it as required. Business Object Designer assumes you name each file according to the naming convention. For example, if the AGENTNAME variable specifies JDBCODA1, the tool assumes that the name of the associated message file is JDBCODA1Agent.txt. Therefore, when Business Object Designer provides the filename for verification as part of ODA configuration, the filename is based on the ODA name. Verify that the default message file is named correctly, and correct it as necessary.

Important: Failing to correctly specify the message file's name when you configure the ODA causes it to run without messages. For more information on specifying the message file name, see "Configure initialization properties" on page 69.

During the configuration process, you specify:

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

Table 13 describes these values.

Table 13. Tracing levels

Trace level	Description
0	Logs all errors
1	Traces all entering and exiting messages for method
2	Traces the ODA's properties and their values
3	Traces the names of all business objects
4	Traces details of all spawned threads
5	• Indicates the ODA initialization values for all of its properties • Traces a detailed status of each thread that JDBCODA spawned • Traces the business object definition dump

For information on where you configure these values, see "Configure initialization properties" on page 69.

Using JDBCODA in business object designer

This section describes how to use JDBCODA in Business Object Designer to generate business object definitions. For information on launching Business Object Designer, see the *Business Object Development Guide*.

After you launch an ODA, you must launch Business Object Designer to configure and run it. There are six steps in Business Object Designer to generate a business object definition using an ODA. Business Object Designer provides a wizard that guides you through each of these steps.

After starting the ODA, do the following to start the wizard:

1. Open Business Object Designer.
2. From the File menu, select the New Using ODA... submenu.
Business Object Designer displays the first window in the wizard, named Select Agent. Figure 6 on page 69 illustrates this window.

To select, configure, and run the ODA, follow these steps:

1. "Select the ODA" on page 69
2. "Configure initialization properties" on page 69
3. "Expanding nodes and selecting tables, views and stored procedures" on page 71
4. "Confirming database object selections" on page 71
5. "Generating definitions" on page 72 and, optionally, "Providing additional information" on page 72

6. "Saving definitions" on page 74

Select the ODA

Figure 6 illustrates the first dialog box in Business Object Designer's six-step wizard. From this window, select the ODA to run.

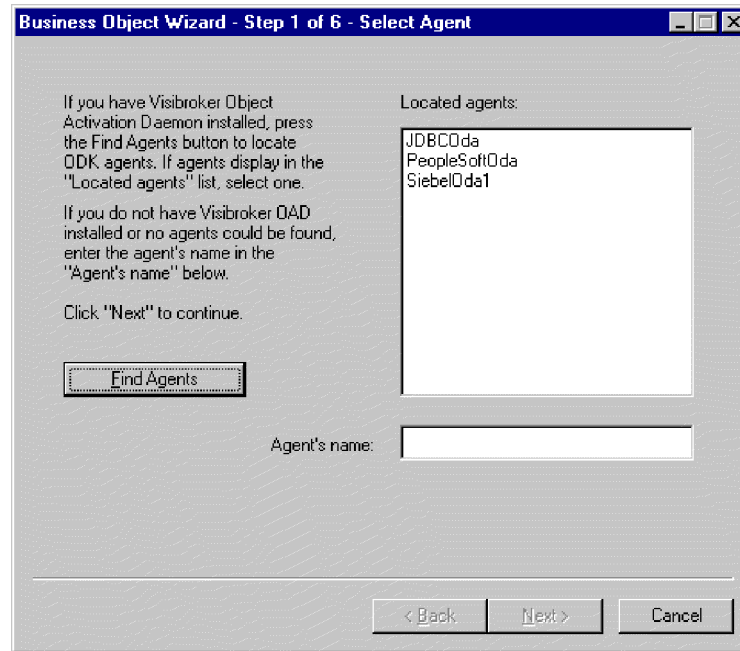


Figure 6. Selecting the ODA

To select the ODA:

1. Click the Find Agents button to display all registered or currently running ODAs in the Located agents field.

Note: If Business Object Designer does not locate your desired ODA, check the setup of the ODA.

2. Select the desired ODA from the displayed list.

Business Object Designer displays your selection in the Agent's name field.

Configure initialization properties

The first time Business Object Designer communicates with JDBCODA, it prompts you to enter a set of initialization properties as shown in Figure 7. You can save these properties in a named profile so that you do not need to re-enter them each time you use JDBCODA. For information on specifying an ODA profile, see the *Business Object Development Guide*.

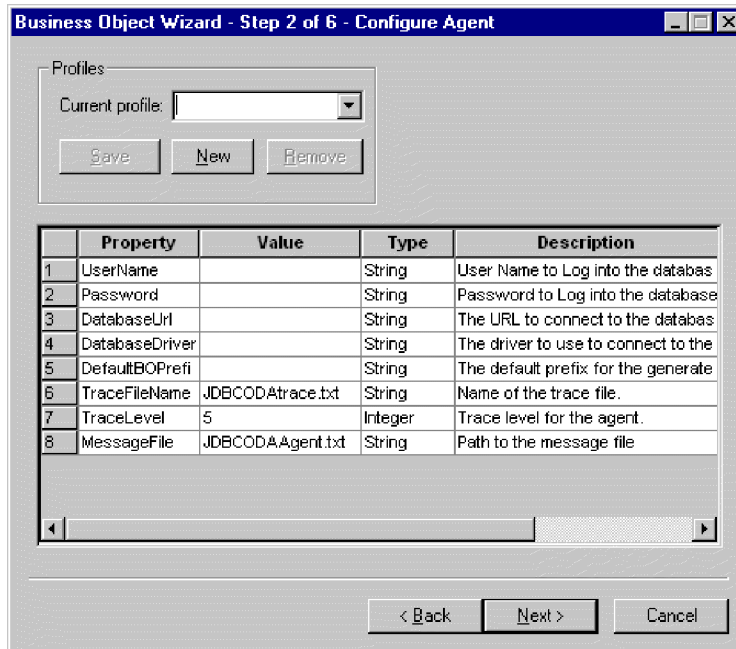


Figure 7. Configuring agent initialization properties

Configure the JDBCODA properties described in Table 14.

Table 14. JDBCODA properties

Row number	Property name	Property type	Description
1	UserName	String	Name of the user with authorization to connect to the database
2	Password	String	Password of the user with authorization to connect to the database
3	DatabaseUrl	String	URL that enables a connection to the database. For example: jdbc:oracle:thin:@MACHINE:1521:SIDNAME
4	DatabaseDriver	String	Name of the driver used to establish the connection. For example: oracle.jdbc.driver.OracleDriver
5	DefaultBOPrefix	String	Text that is prepended to the name of the business object to make it unique. You can change this later, if required, when Business Object Designer prompts you for business object properties. For more information, see "Providing additional information" on page 72.
6	TraceFileName	String	File into which JDBCODA writes trace information. If the file does not exist, JDBCODA creates it in the \ODA\JDBC directory. If the file already exists, JDBCODA appends to it. JDBCODA names the file according to the naming convention. For example, if the agent is named JDBCODA, it generates a trace file named JDBCODATrace.txt. Use this property to specify a different name for this file.
7	TraceLevel	Integer	Level of tracing enabled for JDBCODA
8	MessageFile	String	Name of the error and message file. JDBCODA displays the filename according to the naming convention. For example, if the agent is named JDBCODA, the value of the message file property displays as JDBCODAAgent.txt. Important: The error and message file must be located in the \ODA\messages directory. Use this property to verify or specify an existing file.

Important

Correct the name of the message file if the default value displayed in Business Object Designer represents a non-existent file. If the name is not correct when you move forward from this dialog box, Business Object Designer displays an error message in the window from which the ODA was launched. This message does not popup in Business Object Designer. Failing to specify a valid message file causes the ODA to run without messages.

Expanding nodes and selecting tables, views and stored procedures

After you configure all initialization properties for JDBCODA, Business Object Designer connects to the specified database and displays a tree with all the schema names in the database. These names, which are presented as nodes in the tree, are expandable. Click on them to display all the tables, views and stored procedures in each schema. Figure 8 illustrates this dialog box with some schema expanded.

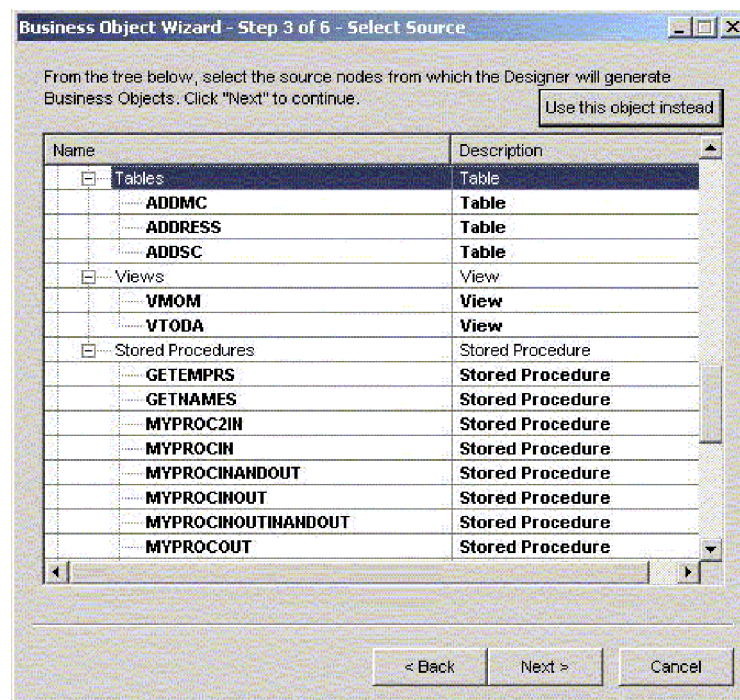


Figure 8. Tree of Schema with Expanded Nodes

To identify all the database objects that store data for the generated business object definition, select all the required tables, views and stored procedures, and click Next. For information on how to filter the objects returned, see the *Business Object Development Guide*.

Confirming database object selections

After you identify all the database objects to be associated with the generated business object definition, Business Object Designer displays the dialog box with only the selected tables and views. Figure 9 illustrates this dialog box.

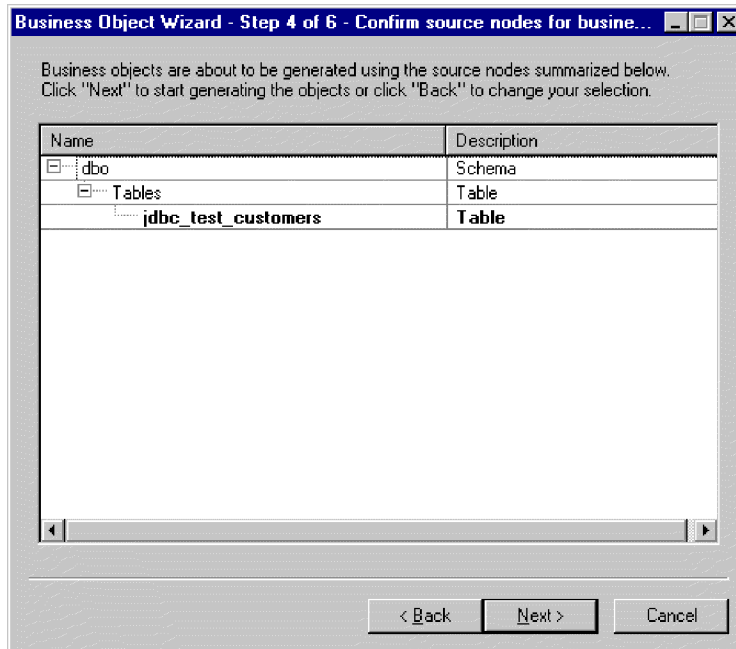


Figure 9. Confirming Selection of Database Objects

This window provides the following options:

- To confirm the selection, click Next.
- If the selection is not correct, click Back to return to the previous window and make the necessary changes. When the selection is correct, click Next.

Generating definitions

After you confirm the database objects, the next dialog box informs you that Business Object Designer is generating the definitions.

Providing additional information

If the JDBCODA needs additional information, Business Object Designer displays the BO Properties window, which prompts you for the information.

In the BO Properties window, enter or change the following information:

- *Prefix*—The text that is prepended to the name of the business object to make it unique. If you are satisfied with the value you entered for the *DefaultBOPrefix* property in the Configure Agent window (Figure 7), you do not need to change the value here.
- *Verbs*— Click in the *Value* field and select one or more verbs from the pop-up menu. These are the verbs supported by the business object.
- *Add Stored Procedure*—Click Yes or No in the Value field:
 - If you select Yes and click OK, JDBCODA displays a window that provides a list of all stored procedure attributes. Select the stored procedure attributes that you want added to the business object.
 - Select No to ensure that no stored procedure attributes are added to the generated business object definition.

The default is Yes.

Note: If a field in the BO Properties dialog box has multiple values, the field appears to be empty when the dialog box first displays. Click in the field to display a drop-down list of its values.

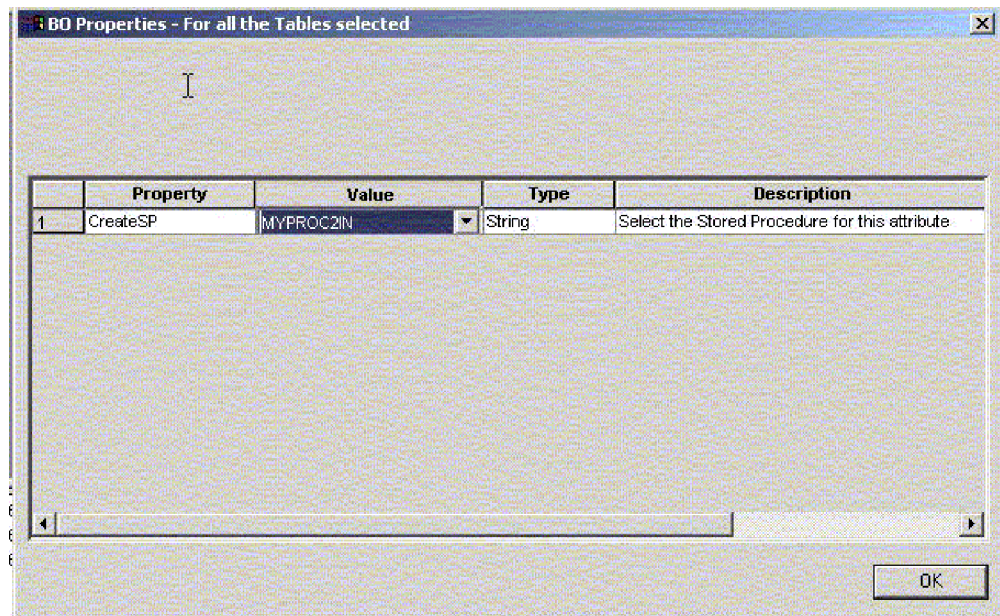


Figure 10. Associating Stored Procedures with Stored Procedure Attributes

The stored procedure attributes to be added to the business object can be associated with one of the stored procedures in the database in that schema. You can choose a stored procedure from a drop down list of all stored procedures in the database in that schema, against each stored procedure attribute. This information will generate the necessary ASI information for that attribute.

The ASI (application-specific information) for the object level will look like
 TN=tableName

And for the attribute level, the ASI will look like CN=ColumnName

If a business object is being generated from a stored procedure, and if JDBC Adapter stored procedure attributes, such as SPForCreate, are associated with it, then the ODA provides a list of all stored procedure names in that schema against the stored procedure attributes and enables you to associate the required stored procedure with the business object. This will generate the ASI for the JDBC Adapter stored procedure attribute as follows:

SPN=stored procedure Name; IN=a1:a2; OUT=b1:b2; IO=c1:c2

Where IN means the parameter of the stored procedure is INPUT type, OUT means the parameter is OUTPUT type, and IO means it is INPUT/OUTPUT type. The ODA will not set RS to true or false on the ASI, so you need to set it manually.

The verbs added to the business object are the standard verbs, essentially Retrieve, RetrieveByContent, Create, Update and Delete.

If the return parameter of the stored procedure is of ResultSet Type, the ODA will analyse the result set and create a business object, making the columns of the result set attributes of the business object. The ASI for the stored procedure columns will

be set as `CN=StoredProcedureColumnName`. The ODA sets the key attributes based on the JDBC metadata information returned by the driver. If none is returned, the ODA does not mark any attributes by default as keys. All other attributes, such as length and type, are set as for the attributes generated from tables.

Saving definitions

After you provide all required information in the BO Properties dialog box and click OK, Business Object Designer displays the final dialog box in the wizard. Here, you can save the definition to the server or to a file, or you can open the definition for editing in Business Object Designer. For more information, and to make further modifications, see the *Business Object Development Guide*.

Contents of the generated definition

The business object definition that JDBCODA generates contains:

- An attribute for each column in the specified database tables and views
- The verbs specified in the BO Properties window
- Application-specific information:
 - At the business-object level
 - For each attribute
 - For each verb

This section describes:

- “Business-object-level properties”
- “Attribute properties” on page 75
- “Verbs” on page 77

Business-object-level properties

JDBCODA generates the following information at the business-object level:

- Name of the business object
- Version—defaults to 1.0.0
- Application-specific information

Application-specific information at the business-object level allows you to:

- Specify the name of the corresponding database table
- Provide the information necessary to perform a physical or logical delete

At the business-object level, application-specific information format consists of parameters separated by semicolon (;) delimiters. The name of the parameter and its value are separated by a colon (:) delimiter. The syntax is:

```
TN=TableName; SCN=StatusColumnName:StatusValue
```

where `TableName` identifies the database table, `StatusColumnName` is the name of the database column used to perform logical deletes, and `StatusValue` is the value that signifies that a business object is inactive or deleted.

The `AppSpecificInfo` that JDBCODA generates at this level contains a value only for the name of the database table or view. For information on specifying a value for the status column, see “Application-specific information at the business-object level” on page 54.

Attribute properties

This section describes the properties that JDBCODA generates for each attribute. For more information about the attributes, see “Business object attribute properties” on page 50.

Name property

JDBCODA obtains the value of the attribute’s name from the column name in the database table or view.

Data type property

When setting the type of an attribute, JDBCODA converts the data type of a column in the table or view to a corresponding IBM WebSphere Business Integration Adapter Business Object type. This conversion is done in two steps. First, the data type in the database is converted to a JDBC type. Then, the JDBC type is converted to an IBM WebSphere Business Integration Adapter Business Object type. The first conversion is done by the JDBC driver that you are using. Please refer to the JDBC specification (2.0 and above) for details on individual database type mapping to a JDBC type. Table 14 shows the conversion from the JDBC Type to the corresponding IBM WebSphere Business Integration Adapter Business Object type.

Table 15. Correspondence of data types

JDBC type	WebSphere Business Integration Adapter business object type
BIT	BOOLEAN
CHAR	STRING
VARCHAR	STRING
LONGVARCHAR	STRING
INTEGER	INTEGER
NUMERIC	INTEGER
SMALLINT	INTEGER
TINYINT	INTEGER
BIGINT	INTEGER
DATE	DATE
TIME	DATE
TIMESTAMP	DATE
DECIMAL	STRING
DOUBLE	DOUBLE
FLOAT	DOUBLE
REAL	FLOAT
BINARY	STRING, add BYTEARRAY=TRUE to AppSpecificInfo

Table 15. Correspondence of data types (continued)

JDBC type	WebSphere Business Integration Adapter business object type
VARBINARY	STRING, add BYTEARRAY=TRUE to AppSpecificInfo

Note: If a column's data type is not one of those shown in Table 15 on page 75, JDBCODA skips the column and displays a message stating that the column cannot be processed.

Cardinality property

JDBCODA sets the cardinality of all simple attributes to 1.

MaxLength property

JDBCODA obtains the length of a string from the length specified for the varchar, char, or text data type.

IsKey property

If the column is a primary key in the table, JDBCODA marks it as a key attribute. However, if a view, instead of a table, is selected as the source node to generate Business Objects, JDBCODA does not mark the column as a key attribute. In this case, the key attribute needs to be set manually.

IsForeignKey property

JDBCODA does not set the IsForeignKey property. You can set it in Business Object Designer.

IsRequired property

If a field is designated not null in the table or view, JDBCODA marks it as a required attribute. However, JDBCODA does not mark the key field as required because there may be a sequence associated with it, or it may be an identity column.

AppSpecificInfo property

JDBCODA includes two parameters for the AppSpecificInfo property at the attribute level. The syntax of the specified parameters are:

CN=ColumnName

where ColumnName is the name of the column in the database table or view associated with the specific attribute.

BYTEARRAY=true|false

JDBCODA recognizes columns with binary data and creates an attribute of type String with an AppSpecificInfo property of BYTEARRAY=true.

Note: You can set additional AppSpecificInfo parameters in Business Object Designer. For information about these parameters, see "Application-specific information at the attribute level" on page 55.

Verbs

JDBCODA generates the verbs specified in the BO Properties window. It creates an AppSpecificInfo property for each verb but does not populate it. For more information, see “Application-specific information format for verbs” on page 62.

Sample business object definition file

A sample business object definition follows:

```
[BusinessObjectDefinition]
Name = CUSTOMER
Version = 1.0.0
AppSpecificInfo = TN=ra_customers;SCN=

    [Attribute]
    Name = customer_id
    Type = Integer
    Cardinality = 1
    MaxLength = 0
    IsKey = true
    IsForeignKey = false
    IsRequired = false
    AppSpecificInfo = CN=customer_id
    DefaultValue =
    [End]

    *****Other attributes *****

[Attribute]
Name = ObjectEventId
Type = String
Cardinality = 1
MaxLength = 0
IsKey = false
IsForeignKey = false
IsRequired = false
AppSpecificInfo =
DefaultValue =
[End]

[Verb]
Name = Delete
AppSpecificInfo =
[End]

[Verb]
Name = Update
AppSpecificInfo =
[End]

[Verb]
Name = Create
AppSpecificInfo =
[End]

[Verb]
Name = Retrieve
AppSpecificInfo =
[End]

[End]
```

Inserting attributes that contain child business objects

Use Business Object Designer to insert attributes that represent single-cardinality or multiple-cardinality child business objects. For more information, see the *Business Object Development Guide*.

Adding information to the business object definition

Because the database tables and views may not have all the information that a business object definition requires, it may be necessary to add information to the business object definition that JDBCODA creates. For more information, see Chapter 3, “Understanding business objects for the connector,” on page 31.

To examine the business object definition or add information, you can use Business Object Designer or a text editor. To reload a revised definition into the IBM WebSphere Business Integration Adapter repository, you can use Business Object Designer or, if ICS is the integration broker, the `repos_copy` command.

Chapter 5. Troubleshooting and error handling

The chapter describes problems that you may encounter when starting up or running the connector for JDBC. It contains the following sections:

- “Startup problems”
- “Event processing”
- “Mapping (ICS Integration Broker only)”
- “Error handling and logging” on page 80
- “Loss of connection to the application” on page 82
- “Inability to locate event or archive tables when DB2 is used” on page 82
- “Resource-busy error” on page 83
- “JDBCODA behaves improperly because of unsupported JDBC driver” on page 83

Startup problems

If you encounter difficulties when trying to start the connector, check to make sure that integration broker is up and running.

Event processing

If there are events in the event table, and they are not being processed while the connector is running, ensure that:

- The relevant business process is running.
- The name of the business object in the event table matches the name of the business object specified for the business process port.

Mapping (ICS Integration Broker only)

This section discusses the following:

- “Mapping problems”
- “Date conversion”

Mapping problems

If the business objects are not being mapped or mapping is not being invoked, check to make sure the maps have been installed in the correct directory.

Date conversion

Note: This date conversion procedure applies only to versions of the connector prior to version 1.5.0.

Use maps to convert data stored in Date format in the database to the String format used by a WebSphere Business Integration Adapter business object.

For example, assume that you want to convert the following date, which is stored in an Oracle database:

```
Sun Jan 01 00:00:00 CEST 1999
```

to the following string, which is processed in a WebSphere Business Integration Adapter for JDBC business object:

```
Jan 01 1999 00:00:00
```

To perform this conversion, use the `DtpDate()` and `DtpSplitString()` constructors defined for data transformation in mapping. For the syntax and a description of these constructors and the classes whose objects they construct, see the *Map Development Guide*.

To use a map to convert the `Date` value to a `String`, follow these steps:

1. Use `DtpSplitString()` with a space delimiter to split the string into its six pieces and rearrange it into an order that `DtpDate` can use. To convert the example date, use:

```
DtpSplitString OurSplitString = new DtpSplitString
("Sun Jan 01 00:00:00 CEST 1999", " ");
```

In the above statement, `OurSplitString` is a user-defined variable of type `DtpSplitString`, and a space is specified as the delimiter.

2. Use the `nextElement()` method of the `DtpSplitString` class to loop through the newly created `OurSplitString` variable, putting each of the variable's six elements into an array whose elements are of type `String`. The following example specifies `OurStringPieces` as the output array:

```
String[] OurStringPieces = new String[6];
for (i=0;i<=5;i=i+1){
    OurStringPieces[i]=OurSplitString.nextElement();
}
```

This looping produces the following array elements:

```
OurStringPieces[0] = Sun
OurStringPieces[1] = Jan
OurStringPieces[2] = 01
OurStringPieces[3] = 00:00:00
OurStringPieces[4] = CEST
OurStringPieces[5] = 1999
```

3. Concatenate the pieces of the string needed for `DtpDate` input. The example conversion uses "M D Y h:m:s" as the input format for `DtpDate`, which requires the converted string to look like "Jan 01 1999 00:00:00". This example `String` uses elements 1, 2, 5, and 3 of the `OurStringPieces` array:

```
OurConcatenatedString =
OurStringPieces[1]+OurStringPieces[2]+OurStringPieces[5]+OurStringPieces[3];
```

4. Use your new concatenated string as input into `DtpDate`:

```
DtpDate OurDtpDate = new DtpDate(OurConcatenatedString,"M D Y h:m:s");
```

After you have put the `Date` value into `DtpDate` format, you are ready to work with the date in your map.

Error handling and logging

The connector logs an error message whenever it encounters a condition that causes its current processing of a business object and verb to fail. When such an error occurs, the connector also prints a textual representation of the failed business object as it was received. It writes the text to the connector log file or the standard output stream, depending on its configuration. You can use the text as an aid in determining the source of the error.

Error types

Table 16 describes the types of tracing messages that the connector outputs at each trace level. These messages are in addition to any tracing messages output by the IBM WebSphere Business Integration Adapter architecture, such as the Java connector execution wrapper and the WebSphere MQ message interface.

Table 16. Connector tracing messages

Tracing level	Tracing messages
Level 0	Message that identifies the connector version. No other tracing is done at this level. This is the default value.
Level 1	<ul style="list-style-type: none"> • Status messages • Messages that provide identifying (key) information for each business object processed • Messages delivered each time the pollForEvents method is executed
Level 2	<ul style="list-style-type: none"> • Business object handler messages that contain information such as the arrays and child business objects that the connector encounters or retrieves during the processing of a business object • Messages logged each time a business object is posted to the integration broker, either from <code>gotApp1Event()</code> or <code>executeCollaboration()</code> • Messages that indicate that a business object has been received as an integration broker request
Level 3	<ul style="list-style-type: none"> • Foreign key processing messages that contain such information as when the connector has found or has set a foreign key in a business object • Messages that provide information about business object processing. For example, these messages are delivered when the connector finds a match between business objects, or finds a business object in an array of child business objects
Level 4	<ul style="list-style-type: none"> • Application-specific information messages, for example, messages showing the values returned by the functions that parse the business object's application-specific information fields • Messages that identify when the connector enters or exits a function, which helps trace the process flow of the connector • All thread-specific messages. If the connector spawns multiple threads, a message appears for the creation of each new thread
Level 5	<ul style="list-style-type: none"> • Messages that indicate connector initialization, for example, messages showing the value of each configuration property retrieved from the integration broker • Messages that include statements executed in the application. At this trace level, the connector log file contains all statements executed in the destination application and the value of any variables that are substituted. • Messages that comprise a representation of a business object before the connector begins processing it (displaying its state as the connector receives it) and after the connector has completed its processing (displaying its state as the connector returns it) • Messages that comprise a business object dump • Messages that indicate the status of each thread the connector spawns while it is running

Error messages

Connector message file

All the error messages that the connector generates are stored in a message file named `JDBCConnector.txt` or `JDBCConnector_II_TT.txt` (where *II* specifies a language, and *TT* specifies a country or territory). Each error has an error number followed by the error message. For example:

20017

Connector Infrastructure version does not match.

20018

Connection from {1} to the Application is lost! Please enter 'q' to stop the connector, then restart it after the problem is fixed.

20019

Error: `ev_id` is NULL in `pollForEvent()`.

Loss of connection to the application

If the connector fails to establish connection, it sends FAIL to the integration broker and terminates.

When `AutoCommit` is set to false and the `PingQuery` fails, the connector will attempt to create a new connection to the database. If it succeeds in creating a new connection to the database it will continue processing, otherwise the connector returns an `APPRESPONSETIMEOUT`, which results in the termination of the connector.

Fetch out-of-sequence error

The `AutoCommit` property must be set to false when using Oracle database version 8.0 and 8.1 with Sun Solaris or Oracle 8.1 with Windows 2000. Otherwise, you will experience ORA-01002 (fetch out of sequence) error messages. In prior versions of Oracle databases this error will not occur. Setting `AutoCommit` to false will improve performance.

Inability to locate event or archive tables when DB2 is used

During startup, the connector attempts to locate the event and archive tables within the database specified by the `SchemaName` configuration property. If you are using DB2 as the database, the connector sometimes fails to locate the event and archive tables and returns the following error:

Event/Archive table `table_name` does not exist in the database.

To avoid this problem, always specify DB2 schema names in upper case (for example, `SUSER`) in the connector's `SchemaName` configuration property.

Enabling the connector to work with a DB2 database

Before you can use the connector with a DB2 database, you must perform the following steps:

1. Copy the file named `db2java.zip` from the DB2 host to the `$ProductDir\lib` directory on the machine on which the connector is going to run.
2. Copy the file named `db2jdbc.dll` from the DB2 host to the `$ProductDir\bin` directory on the machine on which the connector is going to run.
3. Depending on your operating system, change the following in the connector's startup file (`start_JDBC.sh` or `start_JDBC.bat`):

UNIX: `JDBC_DRIVER_PATH=$ProductDir/lib/db2java.zip`

Windows: set JDBC_DRIVER_PATH=%ProductDir%\lib\db2java.zip

4. On the DB2 host machine, start the DB2/bin/db2jstrt process. Be sure to specify the number of the port you are using (for example, DB2/bin/db2jstrt 50000).
5. Set the value of the connector's JDBC_DRIVER_CLASS property to COM.ibm.db2.jdbc.net.DB2Driver (or COM.ibm.db2.jdbc.app.DB2Driver if the DB2 database is on the same machine on which the connector is going to run).
6. Set the value of the connector's DATABASE_URL property to jdbc:db2://MachineName:PortNumber/DBname (or jdbc:db2:DBname if the DB2 database is on the same machine on which the connector is going to run).

Resource-busy error

Note: This connector only encounters this error when it is running on an Oracle database.

The connector sometimes encounters an error like the following when retrieving or changing data in an application.

```
[Time: 2001/05/29 16:30:07.356] [System: ConnectorAgent] [SS: SOVTConnector]
[Type: Trace] [Msg: Select CLIENT,COUNTRY,STRT_CODE,CITY_CODE,CITYP_CODE,
STRTYPEAB,COMMU_CODE,REGIOGROUP,TAXJURCODE from ADRSTREET where CLIENT='100'
and COUNTRY='DE' and STRT_CODE='000001114136' FOR UPDATE NOWAIT]
[Time: 2001/05/29 16:30:07.526] [System: ConnectorAgent] [SS: SOVTConnector]
[Type: Trace ] [Msg: :logMsg]
[Time: 2001/05/29 16:30:07.536] [System: ConnectorAgent] [SS: SOVTConnector]
[Type: Error ] [MsgID: 37002]
[Msg: Execution of Retrieve statement failed : java.
sql.SQLException: ORA-00054: Versuch, mit NOWAIT eine bereits
belegte Ressourcenzufordern.]
```

This error occurs when the connector tries to update a record that is currently locked. The record may be locked by another process, or because the connector is multi-threaded, it may be locked by the connector itself.

Note that records must be locked during the update process. The connector attempts to retrieve an afterimage of the object received by the integration broker and, in the process, locks the entire object in the database to preserve data integrity.

To resolve this problem, you can stop the process that is preventing the connector from obtaining a lock on the record, or you can adjust the RetryCountInterval configuration property for the connector.

JDBCODA behaves improperly because of unsupported JDBC driver

If the JDBC driver does not support a feature of JDBCODA, the object discovery agent does not function properly. For example, if the driver does not support all method calls that JDBCODA uses, the JDBCODA log indicates the failed process. The following is an example from the log:

```
[Time: 2002/05/15 17:00:55.147] [System: Object Discovery Agent] [SS: null]
[Type: 6] [Msg: A SQL Error occurred in getting Schema Names from Database.
Reason [ProductName][ODBC ProductName Driver]Optional feature not
implemented]
```

In such a case, you must use a different JDBC driver.

Appendix A. Standard configuration properties for connectors

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

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

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

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

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

New and deleted properties

These standard properties have been added in this release.

New properties

- XMLNamespaceFormat

Deleted properties

- RestartCount

Configuring standard connector properties

Adapter connectors have two types of configuration properties:

- Standard configuration properties
- Connector-specific configuration properties

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

Using Connector Configurator

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

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

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

Setting and updating property values

The default length of a property field is 255 characters.

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

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

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

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

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

Summary of standard properties

Table 17 on page 87 provides a quick reference to the standard connector configuration properties. Not all the connectors make use of all these properties, and property settings may differ from integration broker to integration broker, as standard property dependencies are based on RepositoryDirectory.

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

Table 17. Summary of standard configuration properties

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

Table 17. Summary of standard configuration properties (continued)

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

Table 17. Summary of standard configuration properties (continued)

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

Table 17. Summary of standard configuration properties (continued)

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

Standard configuration properties

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

AdminInQueue

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

The default value is CONNECTORNAME/ADMININQUEUE.

AdminOutQueue

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

The default value is CONNECTORNAME/ADMINOUTQUEUE.

AgentConnections

Applicable only if RepositoryDirectory is <REMOTE>.

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

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

AgentTraceLevel

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

ApplicationName

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

BrokerType

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

CharacterEncoding

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

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

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

ConcurrentEventTriggeredFlows

Applicable only if `RepositoryDirectory` is `<REMOTE>`.

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

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

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

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

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

ContainerManagedEvents

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

The default value is `No value`.

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

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

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

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

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

ControllerStoreAndForwardMode

Applicable only if `RepositoryDirectory` is `<REMOTE>`.

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

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

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

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

The default is `true`.

ControllerTraceLevel

Applicable only if `RepositoryDirectory` is `<REMOTE>`.

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

DeliveryQueue

Applicable only if DeliveryTransport is JMS.

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

The default value is CONNECTORNAME/DELIVERYQUEUE.

DeliveryTransport

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

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

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

WebSphere MQ and IDL

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

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

JMS

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

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

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

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

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

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

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


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

DuplicateEventElimination

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

This property can also be set to false.

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

FaultQueue

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

The default value is `CONNECTORNAME/FAULTQUEUE`.

JvmMaxHeapSize

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

The default value is 128m.

JvmMaxNativeStackSize

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

The default value is 128k.

JvmMinHeapSize

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

The default value is 1m.

jms.FactoryClassName

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

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

jms.MessageBrokerName

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

The default is `crossworlds.queue.manager`.

jms.NumConcurrentRequests

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

The default value is 10.

jms.Password

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

There is no default.

jms.UserName

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

There is no default.

ListenerConcurrency

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

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

Locale

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

A locale name has the following format:

ll_TT.codeset

where:

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

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

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

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

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

LogAtInterchangeEnd

Applicable only if `RepositoryDirectory` is `<REMOTE>`.

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

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

MaxEventCapacity

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

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

MessageFileName

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

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

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

MonitorQueue

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

The default value is CONNECTORNAME/MONITORQUEUE

OADAutoRestartAgent

Valid only when the RepositoryDirectory is <REMOTE>.

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

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

The default value is false.

OADMaxNumRetry

Valid only when the RepositoryDirectory is <REMOTE>.

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

The default value is 1000.

OADRetryTimeInterval

Valid only when the RepositoryDirectory is <REMOTE>.

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

The default is 10.

PollEndTime

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

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

PollFrequency

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

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

The default is 10000.

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

PollQuantity

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

PollStartTime

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

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

RequestQueue

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

The default value is `CONNECTOR/REQUESTQUEUE`.

RepositoryDirectory

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

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

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

ResponseQueue

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

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

RestartRetryCount

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

The default is 3.

RestartRetryInterval

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

The default is 1.

RHF2MessageDomain

WebSphere message brokers and WAS only.

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

A sample header would look like this:

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

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

SourceQueue

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

Designates the JMS source queue for the connector framework in support of guaranteed event delivery for JMS-enabled connectors that use a JMS event store. For further information, see “`ContainerManagedEvents`” on page 92.

The default value is `CONNECTOR/SOURCEQUEUE`.

SynchronousRequestQueue

Applicable only if `DeliveryTransport` is `JMS`.

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

The default is `CONNECTORNAME/SYNCHRONOUSREQUESTQUEUE`

SynchronousResponseQueue

Applicable only if `DeliveryTransport` is `JMS`.

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

The default is CONNECTORNAME/SYNCHRONOUSRESPONSEQUEUE

SynchronousRequestTimeout

Applicable only if DeliveryTransport is JMS.

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

The default value is 0.

WireFormat

Message format on the transport.

- If the RepositoryDirectory is a local directory, the setting is CwXML.
- If the value of RepositoryDirectory is <REMOTE>, the setting is CwB0.

WsifSynchronousRequest Timeout

WAS integration broker only.

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

The default value is 0.

XMLNameSpaceFormat

WebSphere message brokers and WAS integration broker only.

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

The default value is short.

Appendix B. Connector Configurator

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

You use Connector Configurator to:

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

Note:

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

The topics covered in this appendix are:

- “Overview of Connector Configurator” on page 101
- “Starting Connector Configurator” on page 102
- “Creating a connector-specific property template” on page 103
- “Creating a new configuration file” on page 105
- “Setting the configuration file properties” on page 108
- “Using Connector Configurator in a globalized environment” on page 114

Overview of Connector Configurator

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

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

You use Connector Configurator to:

- Create a **connector-specific property template** for configuring your connector.
- Create a **connector configuration file**; you must create one configuration file for each connector you install.
- Set properties in a configuration file.

You may need to modify the default values that are set for properties in the connector templates. You must also designate supported business object definitions and, with ICS, maps for use with collaborations as well as specify messaging, logging and tracing, and data handler parameters, as required.

The mode in which you run Connector Configurator, and the configuration file type you use, may differ according to which integration broker you are running. For example, if WMQI is your broker, you run Connector Configurator directly, and not from within System Manager (see “Running Configurator in stand-alone mode” on page 102).

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 103 to set up a new one.

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

Starting Connector Configurator

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

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

Running Configurator in stand-alone mode

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

To do so:

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

You may choose to run Connector Configurator independently to generate the file, and then connect to System Manager to save it in a System Manager project (see “Completing a configuration file” on page 107.)

Running Configurator from System Manager

You can run Connector Configurator from System Manager.

To run Connector Configurator:

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

To edit an existing configuration file:

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

Creating a connector-specific property template

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

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

- To create a new template, see “Creating a new template” on page 103.
- To use an existing file, simply modify an existing template and save it under the new name.

Creating a new template

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

To create a template:

1. Click **File>New>Connector-Specific Property Template**.
2. The **Connector-Specific Property Template** dialog box appears, with the following fields:
 - **Template, and Name**
Enter a unique name that identifies the connector, or type of connector, for which this template will be used. You will see this name again when you open the dialog box for creating a new configuration file from a template.
 - **Old Template, and Select the Existing Template to Modify**
The names of all currently available templates are displayed in the Template Name display.

- To see the connector-specific property definitions in any template, select that template's name in the **Template Name** display. A list of the property definitions contained in that template will appear in the **Template Preview** display. You can use an existing template whose property definitions are similar to those required by your connector as a starting point for your template.
3. Select a template from the **Template Name** display, enter that template name in the **Find Name** field (or highlight your selection in **Template Name**), and click **Next**.

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

Specifying general characteristics

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

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

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

Specifying values

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

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

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

Setting dependencies

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

A dependent property is a property that is included in the template and used in the configuration file *only if* the value of another property meets a specific condition. For example, `PollQuantity` appears in the template only if JMS is the transport mechanism and `DuplicateEventElimination` is set to `True`.

To designate a property as dependent and to set the condition upon which it depends, do this:

1. In the **Available Properties** display, select the property that will be made dependent.
2. In the **Select Property** field, use the drop-down menu to select the property that will hold the conditional value.
3. In the **Condition Operator** field, select one of the following:
 - == (equal to)
 - != (not equal to)
 - > (greater than)
 - < (less than)
 - >= (greater than or equal to)
 - <=(less than or equal to)
4. In the **Conditional Value** field, enter the value that is required in order for the dependent property to be included in the template.
5. With the dependent property highlighted in the **Available Properties** display, click an arrow to move it to the **Dependent Property** display.
6. Click **Finish**. Connector Configurator stores the information you have entered as an XML document, under `\data\app` in the `\bin` directory where you have installed Connector Configurator.

Creating a new configuration file

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

To select a broker:

- In the Connector Configurator home menu, click **File>New>Connector Configuration**. The **New Connector** dialog box appears.
- In the **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.

You can also do this:

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

Creating a configuration file from a connector-specific template

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

1. Click **File>New>Connector Configuration**.
2. The **New Connector** dialog box appears, with the following fields:
 - **Name**
Enter the name of the connector. Names are case-sensitive. The name you enter must be unique, and must be consistent with the file name for a connector that is installed on the system.

Important: Connector Configurator does not check the spelling of the name that you enter. You must ensure that the name is correct.
 - **System Connectivity**
Click ICS or WebSphere Message Brokers or WAS.
 - **Select Connector-Specific Property Template**
Type the name of the template that has been designed for your connector. The available templates are shown in the **Template Name** display. When you select a name in the Template Name display, the **Property Template Preview** display shows the connector-specific properties that have been defined in that template.
Select the template you want to use and click **OK**.
3. A configuration screen appears for the connector that you are configuring. The title bar shows the integration broker and connector names. You can fill in all the field values to complete the definition now, or you can save the file and complete the fields later.
4. To save the file, click **File>Save>To File** or **File>Save>To Project**. To save to a project, System Manager must be running.
If you save as a file, the **Save File Connector** dialog box appears. Choose *.cfg as the file type, verify in the File Name field that the name is spelled correctly and has the correct case, navigate to the directory where you want to locate the file, and click **Save**. The status display in the message panel of Connector Configurator indicates that the configuration file was successfully created.

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

Using an existing file

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

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

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

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

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

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

1. In Connector Configurator, click **File>Open>From File**.
2. In the **Open File Connector** dialog box, select one of the following file types to see the available files:
 - Configuration (*.cfg)
 - ICS Repository (*.in, *.out)
Choose this option if a repository file was used to configure the connector in an ICS environment. A repository file may include multiple connector definitions, all of which will appear when you open the file.
 - All files (*.*)
Choose this option if a *.txt file was delivered in the adapter package for the connector, or if a definition file is available under another extension.
3. In the directory display, navigate to the appropriate connector definition file, select it, and click **Open**.

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

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

Completing a configuration file

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

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

1. Under the **Standard Properties** tab, select the value field for the BrokerType property. In the drop-down menu, select the value ICS, WMQI, or WAS.
2. The Standard Properties tab will display the properties associated with the selected broker. You can save the file now or complete the remaining configuration fields, as described in “Specifying supported business object definitions” on page 110..
3. When you have finished your configuration, click **File>Save>To Project** or **File>Save>To File**.

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

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

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

Setting the configuration file properties

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

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

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

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

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

- Associated Maps
- Resources
- Messaging (where applicable)

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

Standard properties differ from connector-specific properties as follows:

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

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

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

Setting standard connector properties

To change the value of a standard property:

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

Setting application-specific configuration properties

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

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

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

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

Encryption for connector properties

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

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

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

Update method

Refer to the descriptions of update methods found in the *Standard configuration properties for connectors* appendix, under “Setting and updating property values” on page 86.

Specifying supported business object definitions

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

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

If ICS is your broker

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

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

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

To delete a business object from the supported list:

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

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

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

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

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

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

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

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

If a WebSphere Message Broker is your broker

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

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

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

If WAS is your broker

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

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

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

Associated maps (ICS only)

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

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

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

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

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

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

- **Business Object Name**

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

- **Associated Maps**

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

- **Explicit**

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

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

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

To explicitly bind a map:

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

Resources (ICS)

The **Resource** tab allows you to set a value that determines whether and to what extent the connector agent will handle multiple processes concurrently, using connector agent parallelism.

Not all connectors support this feature. If you are running a connector agent that was designed in Java to be multi-threaded, you are advised not to use this feature, since it is usually more efficient to use multiple threads than multiple processes.

Messaging (ICS)

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

Setting trace/log file values

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

To change the logging and tracing values:

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

Note: You can only use the STDOUT option from the **Trace/Log Files** tab for connectors running on the Windows platform.

- To File:
Writes logging or tracing messages to a file that you specify. To specify the file, click the directory button (ellipsis), navigate to the preferred location, provide a file name, and click **Save**. Logging or tracing message are written to the file and location that you specify.

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

Data handlers

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

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

Saving your configuration file

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

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

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

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

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

Changing a configuration file

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

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

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

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

Completing the configuration

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

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

Using Connector Configurator in a globalized environment

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

Connector Configurator supports non-English characters in:

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

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

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


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

Appendix C. Business object samples

This appendix details the sample business objects that are included with the connector for JDBC. The JDBC connector includes the following business object samples:

- AfterUpdateSPSampleBO.txt
- BeforeCreateSPSampleBO.txt
- BOwithDifferentParameterOrder.txt
- BOwithIOandOPParams.txt
- BOwithFewerSPParamsthanBOAttribs.txt
- CreateSPUpdateSPSampleBO.txt

AfterUpdateSPSampleBO.txt

Attribute name: AfterUpdateSP

Contains the Jdbctest_Customer business object. Application-specific information of this attribute contains the following information:

SPN=UpdateAllColumns;IP=fid:CustomerName:CustomerNumber:CustomerDesc. UpdateAllColumns is the name of the stored procedure that uses all four business object attributes (fid, CustomerName, CustomerNumber and CustomerDesc) as input parameters. This stored procedure is executed after the Update operation is completed.

BeforeCreateSPSampleBO.txt

Attribute name: BeforeCreateSP

Contains the Jdbctest_Customer business object. Application-specific information of this attribute contains the following information: SPN=GetCustomerID;OP=fid

GetCustomerID is the name of the stored procedure that uses the fid business object attribute as an output parameter (preferably to get an ID value from a MasterID table). The stored procedure is executed before the Create operation is completed.

BOwithDifferentParameterOrder.txt

Attribute name: AfterRetrieveSP

Contains the Jdbctest_Address business object. Application-specific information of this attribute contains the following information:

SPN=UpdateAddress;IP=addressid;IP=zipcode:city:street. UpdateAddress is the name of the stored procedure that uses all the business object attributes as input parameters (preferably to update the address in a table other than Jdbctest_Address). Note that the parameter order differs from that of the business object attribute order and that it contains more than one name:value pair for the input parameters. The stored procedure is executed after the Retrieve operation is completed.

BOwithIOandOPParams.txt

Attribute name:RetireveSP

Contains the Jdbctest_Address business object. Application- specific information of this attribute contains the following information:

SPN=RetrieveAddress;IO=addressid;OP=street:city:zipcode. RetrieveAddress is the name of the stored procedure that uses the business object attribute addressid as an input/output parameter. It also uses the remaining business object attributes as output parameters: zipcode, city, street . The stored procedure is executed instead of the Retrieve operation.

BOwithFewerSPParamsthanBOAttribs.txt

Attribute name:AfterUpdateSP

Contains the Jdbctest_Address business object. Application- specific information of this attribute contains the following information:

SPN=UpdateZipOnly;IP=addressid:zipcode. UpdateZipOnly is the name of the stored procedure that uses the addressid and zipcode business object attributes as input parameters. Note that the total number of stored procedure parameters is less than the total number of business object attributes.

CreateSPUpdateSPSampleBO.txt

Attribute name:CreateSP

Contains the Jdbctest_Address business object. Application- specific information of this attribute contains the following information:

SPN=CreateAddress;IP=addressid;IP=street:city:zipcode. CreateAddress is the name of the stored procedure that uses all four business object attributes as input parameters. Note that it contains more than one name:value pair for the input parameters. The stored procedure is executed instead of the Create operation. This business object also contains the UpdateSP attribute. It contains the following text: SPN=UpdateCity;IP=addressid:city. UpdateCity is the name of the stored procedure that uses addressid and city as input parameters. Note that the total number of stored procedure parameters is less than the total number of business attributes. The stored procedure is executed instead of the Update operation.

Appendix D. Support for null and blank values

This appendix details different pass and fail scenarios where the key value in a business object is blank or null. This appendix also contains the functional changes required for blank or null business object values.

Pass and Fail Scenarios

If a key value in a business object is blank or has a null value in the database, then build the where clause with the "is null" type instead of the "=" operator type.

IBM recommends that business objects have at least one key attribute that does not have a blank value.

The following scenario is a parent object with one key that has a null value. This scenario fails under these conditions.

Table 18. Customer

Attribute	Type
cid	Integer (Key)
name	String
comments	String

The following scenario is a parent object with two keys and one key has a null value. This scenario passes under these conditions.

Table 19. Customer

Attribute	Type
cid	Integer (Key)
name	String
comments	String

In scenario two, build the retrieve query by selecting cid, name, and comments from customer, where cid=1000 and name is set to null.

The following scenario is a parent object with one child object in a container object with a foreign key reference. This scenario fails under these conditions.

Table 20. Customer

Attribute	Type
cid	Integer (Key)
name	String (Key)
comments	String
Address	Address
Aid	Integer (Key) ASI:FK=cid
Acity	String

Table 20. Customer (continued)

Attribute	Type
Azip	String

If cid contains a null value, then build the retrieve query by selecting Aid, Acity, and Azip from address. Set the value of Aid to null.

The following scenario is a parent object with one child object in a container object with two key references. This scenario passes under these conditions.

Table 21. Customer

Attribute	Type
cid	Integer (Key)
name	String
comments	String
Address	Address
Aid	Integer (Key) ASI:FK=cid
Acity	String (Key) ASI:FK=name
Azip	String

If name has a null value, then build the Retrieve query by selecting Aid, Acity, and Azip from address, where Aid=Cid and Acity has a null value.

Functionality

If the connector encounters a blank value on a key, it then compares that value with the UseNull value in the attribute. If the value is true, then the connector adds null value to the query. This affects the following verb operations:

- Retrieve
- RetrieveBy Content
- Update
- Delete

Notices

IBM may not offer the products, services, or features discussed in this document in all countries. Consult your local IBM representative for information on the products and services currently available in your area. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any non-IBM product, program, or service.

IBM may have patents or pending patent applications covering subject matter described in this document. The furnishing of this document does not give you any license to these patents. You can send license inquiries, in writing, to:

IBM Director of Licensing
IBM Corporation
North Castle Drive
Armonk, NY 10504-1785
U.S.A.

The following paragraph does not apply to the United Kingdom or any other country where such provisions are inconsistent with local law:

INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Some states do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement may not apply to you.

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make improvements and/or changes in the product(s) and/or program(s) described in this publication at any time without notice.

Any references in this information to non-IBM Web sites are provided for convenience only and do not in any manner serve as an endorsement of those Web sites. The materials at those Web sites are not part of the materials for this IBM product and use of those Web sites is at your own risk.

IBM may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation to you.

Licensees of this program who wish to have information about it for the purpose of enabling: (i) the exchange of information between independently created programs and other programs (including this one) and (ii) the mutual use of the information which has been exchanged, should contact:

IBM Burlingame Laboratory Director
IBM Burlingame Laboratory
577 Airport Blvd., Suite 800

Burlingame, CA 94010
U.S.A

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

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

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

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

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

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

Programming interface information

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

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

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

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

Trademarks and service marks

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

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

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

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

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

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



IBM WebSphere Business Integration Adapter Framework V2.4.



Printed in USA