IBM WebSphere Business Integration Adapters

IBM WebSphere Business Integration Adapter for Portal Infranet User Guide

Version 44.x

IBM WebSphere Business Integration Adapters

IBM WebSphere Business Integration Adapter for Portal Infranet User Guide

Version 44.x

Note!
Before using this information and the product it supports, read the information in "Notices" on page 115.

13September2005

This edition of this document applies to IBM WebSphere Business Integration Adapter for Portal Infranet (5724-G99), version 4.4.0.

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.

Contents

About this document

The IBM[R] WebSphere[R] 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 integration.

What this document includes

This document describes installation, connector property configuration, business object development, and troubleshooting for this IBM WebSphere Business Integration adapter.

What this document does not include

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

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

Audience

This document is for WebSphere consultants and customers who are implementing the connector as part of a WebSphere business-integration system. To use the information in this document, you should be knowledgeable in the following areas:

- Connector development
- Business object development
- Portal Infranet application architecture

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 information center:
  http://www.ibm.com/websphere/integration/wbiadapters/infocenter
- For using adapters with WebSphere InterChange Server, see the IBM WebSphere InterChange Server information centers:
  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:

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

New in this release

Version 4.4.x

Updated in September 2005. The release of this document for adapter version 4.4.x contains the following new or corrected information.

Added support for the management of the Portal Infranet adapter by the IBM Tivoli License Manager (ITLM).

Error handling has been improved to capture more detailed application-specific error information.

Support for IBM WebSphere Business Integration Adapter Framework V2.6.0 has been added.

Support for the following platforms has been added:

- Windows 2000 SP6
- Solaris 9
- HP-UX 11i v1

Support for Sun JDK 1.4.2 has been added.

Support for IBM JDK 1.4.2 has been added.

Support for Oracle9i Release 2 9.2.0.3 and greater has been added.

Prior releases

New in prior releases:

Version 4.3.x

The changes made in version 4.3.x of this connector to do not affect the content of this document.

Version 4.2.x

The adapter for Portal Infranet is now supported on HP-UX 11i.

Beginning with the 4.2x version, the adapter for Portal Infranet is no longer supported on Microsoft Windows NT.

Adapter installation information has been moved from this guide. See Chapter 2 “Installing the Portal Infranet adapter and other files” on page 11 for the new location of that information.

Version 4.1.x

“A complete sample Portal Infranet business object definition” on page 45has been added to provide a model for developing custom business objects.

The adapter can now use WebSphere Application Server as an integration broker. For further information, see “Adapter for Portal Infranet environment” on page 9. The adapter now runs on the following platforms:

- Solaris 7, 8
- AIX 5.x

Version 4.0.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 connector has replaced the CWSAPGEN utility with PORTALODA. For more information, see Chapter 4, “Generating business object definitions using PortalODA,” on page 49.

Version 3.1.x

The internationalized connector is delivered with IBM WebSphere Business Integration Adapter for Portal Infranet.

Version 3.0.x

This release of the connector contains the following new features: The connector has been internationalized. For more information, see “Processing locale-dependent data” on page 8 and Appendix A, “Standard connector properties,” on page 63. The connector supports the following software on AIX 4.3.3:

- Oracle 8.1.7 and Portal Infranet 6.2 SP1
- DB2 7.1.0 and Portal Infranet 6.2 SP1

Version 2.5.x

The IBM WebSphere Business Integration Adapter for Portal Infranet includes the connector for Portal Infranet. This adapter operates with both the InterChange Server (ICS) and WebSphere MQ Integrator integration brokers. An integration broker, which is an application that performs integration of heterogeneous sets of applications, provides services that include data routing. This adapter includes:

- An application-component specific to Portal Infranet
- A sample business object
- IBM WebSphere Adapter Framework, which consists of:
  - Development tools (including Business Object Designer and Connector Configurator)
  - APIs (including ODK, JCDK, and CDK)

This manual provides information about using this adapter with both integration brokers: InterChange Server (ICS) and WebSphere MQ Integrator.

Note: The connector now supports Portal Infranet 6.2.0.

Important: 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.

Version 2.4.x

The changes made in version 2.4.x of this connector to do not affect the content of this document.

Version 2.3.x

Minor changes were made to fix defects and to provide compatibility with IBM CrossWorlds infrastructure version 4.0.0.

Version 2.2.x

The connector version now supports Portal Infranet 6.1.0.

Version 2.1.x

- The connector version now supports Portal Infranet 6.0.1.
- The connector can be installed and run on a UNIX system.
- This document has been substantially reorganized and rewritten.

Chapter 1. Overview of the connector

This chapter provides an overview of the connector component of the IBM WebSphere Business Integration Adapter for Portal Infranet and includes the following sections:

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. The connector framework provides the following services between the integration broker and the application-specific component:

- Receives and sends business objects
- Manages the exchange of startup and administrative messages

This document contains information about the connector framework and the application-specific component, which it refers to as the connector.

The connector enables WebSphere MQ Integrator Broker or IBM WebSphere InterChange Server (ICS) to communicate with Portal Infranet through the exchange of business objects. The Infranet application is a set of software programs developed by Portal Infranet software for managing customer accounts. Customer information, such as account numbers and billing information, is stored in the Infranet database.

The connector and the Portal Infranet application communicate using the Infranet socket-based API. The connector handles transactions using the functions provided by the Infranet API, and the Infranet application notifies the integration broker (WebSphere MQ Integrator Broker or ICS) through the event module when changes occur.

Connector components

The connector for Portal Infranet includes the following components:

- Connector: a Java `.jar` file that implements the business object verb support and the event polling mechanism.
- WebSphere Business Integration Adapter event facilities module: a C++ DLL on Windows and SO files on UNIX, that implements the event notification mechanism in the Infranet application. This module selects the Infranet events relevant to the integration broker and stores them in a table in the Infranet database. The connector polls this table regularly.

The connector generates business objects that it sends to the integration broker. The connector also responds to business object requests from the integration broker. It generates logging and tracing messages that it writes to a file or the connector console, or sends to the integration broker.

Figure 1 illustrates the architecture of the connector and its event mechanism in the Infranet application.

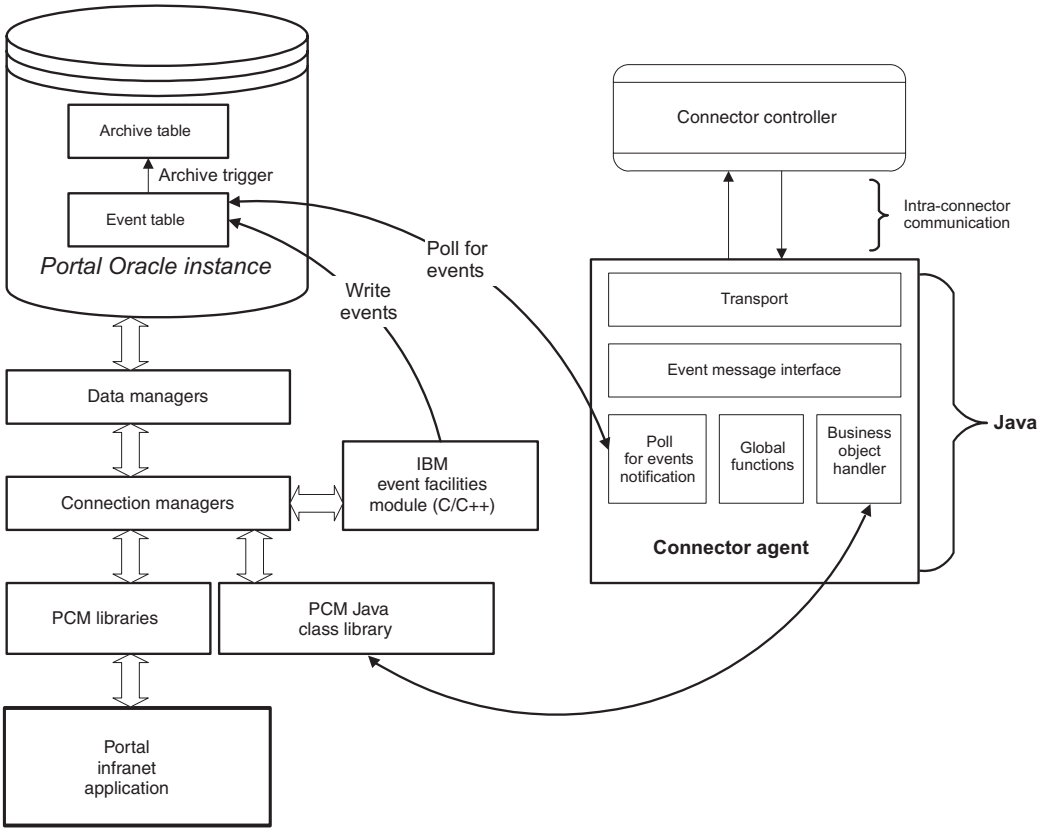


Figure 1. Connector architecture

The connector uses the Portal Communications Module (PCM) Java Class library as the API to interact with the Portal Infranet connection managers. The advantage of this architecture is that Portal Infranet supports the PCM Java class library on the Java virtual machines for both Windows and UNIX. This enables the connector to run on both platforms. This API is used by the business object handler in the connector to exchange information between the integration broker and Portal. The WebSphere Business Integration Adapter event facilities module uses the C++ PCM Library.

How the connector works

The following sections describe how the connector processes business object requests and describe how the connector handles event notification.

Meta-data-driven connector behavior

The connector is meta-data driven. It is designed to handle the retrieval and submission of any business object regardless of the type of business object or the

variables it carries. For the connector to be meta-data driven, business objects for Portal Infranet must contain the following information:

- The field name for each attribute as identified by the data dictionary in Infranet. This includes a pin field number for each pin field name as described in the API documentation. The field name is specified as application-specific information at the attribute level, and the name is converted to the number by the connector using the data dictionary.
- The opcodes (operation codes) that are supported by this business object. The opcodes are specified at the verb level of the business object. An Infranet opcode is an operation used by client applications and scripts to manage customer-related information, create online accounts, collect and track customer information, and integrate third-party systems with Infranet.

For more information on business object meta-data for Portal, see Chapter 3, "Understanding business objects," on page 25.

Business object processing

When the connector receives a business object request from the WebSphere business integration system, the connector business object handler processes the business object. The business object handler is the bridge between the application-specific object and the Portal Infranet API. It is responsible for submitting a Portal Infranet operation to the API and for creating an application-specific business object that is sent to the WebSphere business integration system as the result of an Infranet event. The business object handler uses the data in the business object and any meta-data to make a call to the Infranet Java API to submit a storable object to Portal. When this operation is complete, a status is returned to the integration broker.

The flowchart in Figure 2 shows at a high level how the business object handler processes business object requests. The business object handler extracts the verb and key attributes from the business object. It uses the verb to determine the function call that is made to handle the business object. In this example, if the verb were an update verb, the UpdateObject function would be called.

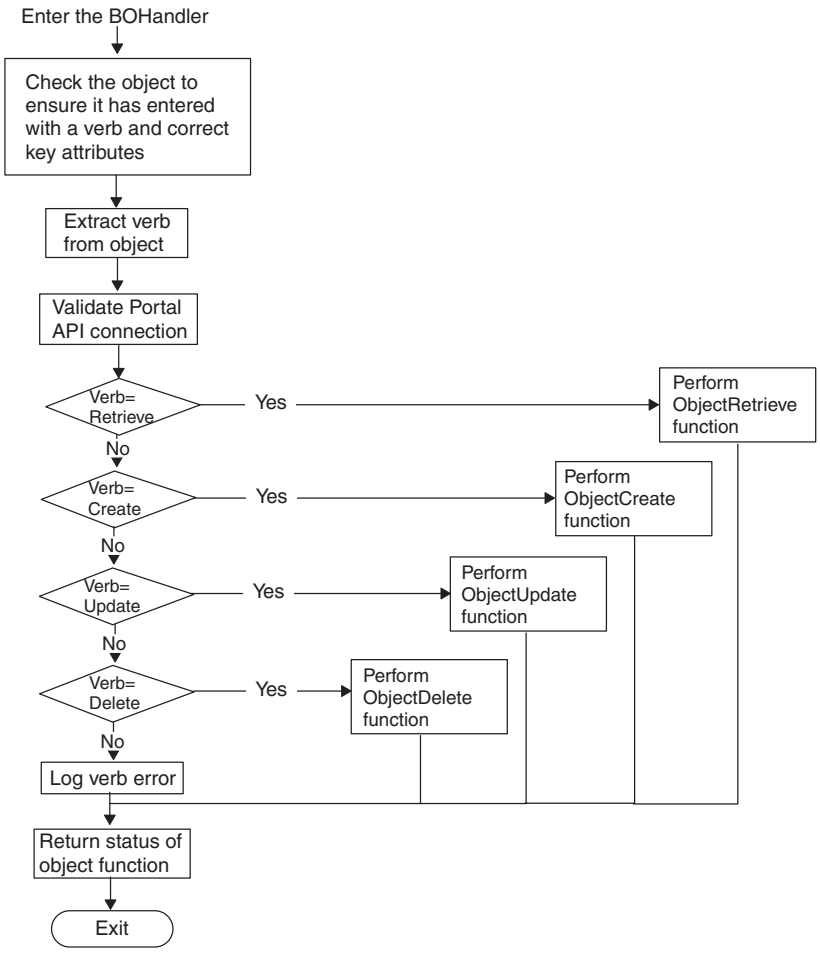


Figure 2. High-level view of business object processing

Retrieve verb processing

The business object handler Retrieve method retrieves an object from Portal Infranet and populates a WebSphere Business Integration Adapter business object with the application information. The connector Retrieve method does the following:

1. Checks that the Portal Infranet connection is valid. If it is not, the connection must be reinstantiated. If the connection is lost during the processing, the connector returns a `BON_FAIL` status indicating a connection problem with Infranet.
2. Retrieves the application-specific information for the object. The application-specific information specified for a verb provides the operation code, or opcode, that must be invoked to retrieve a Portal Infranet object.
3. Prepares the flist for the opcode based on the application-specific information for the business object and its attributes.

   Note: An `flist` is a variable length list of field and value pairs. `Flists` provide input and output parameters to Infranet opcodes and functions.
4. Invokes the specified opcode with the `flist` as input, and an empty output `flist`.

5. If the previous steps are successful, the return `flist` structure contains a fully populated `flist` object corresponding to the storable object that is defined by a WebSphere Business Integration Adapter business object. Since the `flist` has a one-to-one correspondence to the WebSphere Business Integration Adapter business object, the `flist` is traversed to retrieve all the attributes of the WebSphere Business Integration Adapter business object based on the attribute level application-specific information that identifies the `flist` field name and type.
6. Populates the business object and sends it to integration broker.

Figure 3 shows how the Retrieve method works. The method determines which action should be performed on an attribute, depending on its type. If it is a basic attribute type (such as a string), the business object handler dynamically populates the field. If the method encounters a child business object, it descends through the object until it reaches the basic attributes of that child business object. Then it cycles through all the basic attributes of the child object before continuing with the basic attributes of the parent object.

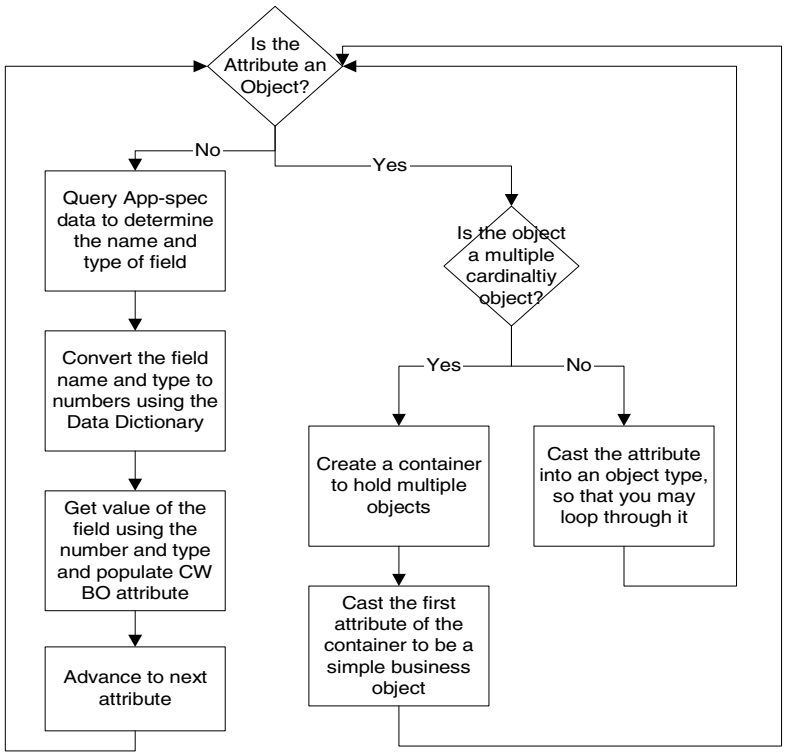


Figure 3. Flowchart for retrieve verb processing

Create and update verb processing

To process a Create or Update verb, the business object handler constructs an Infranet API object (an flist) and passes it to the Infranet API. The business object handler performs the following steps:

1. Retrieves the application-specific business object from the WebSphere Business Integration Adapter business object.
2. Populates the Portal Infranet API object. When the flist is instantiated, the connector iterates through the object, attribute by attribute, and locates the values with which to populate the flist. This process must search through an object to find an attribute that may be several layers into the object.

3. Checks that the Portal Infranet connection is still valid. If the connection is lost during processing, the connector returns a `BON_FAIL` status, indicating a connection problem with Infranet, and the connection must be reinstantiated.
4. Uses the application-specific information for the business object verb to dynamically call the appropriate opcode for the Create or Update operation. Once the parameters are gathered and placed in an array, and the functional string has been assembled, the context function with the appropriate opcode is invoked.
5. To process child business objects that have their own opcodes, the business object handler populates a separate flist for each child and then calls the appropriate opcode.
6. Returns a status when the operation is complete.

Event notification

Infranet has an event mechanism that permits it to keep track of the actions that occur in the application. When a user performs an action in Infranet, the application generates an associated event.

The WebSphere Business Integration Adapter event module examines the event and determines whether it is one that the connector is interested in. If so, the event module generates an entry in the WebSphere Business Integration Adapter event table for the event.

Event detection in infranet

Event detection in Infranet is implemented by means of a custom Infranet facilities module that is called by the event notification mechanism. This facilities module is provided by the integration broker, and it works with two configuration files to identify Infranet events and write events to the WebSphere Business Integration Adapter event table.

When a change occurs to an Infranet object, a persistent event is raised. Infranet can be configured to call a specific opcode when a given event occurs; therefore, the integration broker provides a configuration flat file that configures Infranet to call the WebSphere Business Integration Adapter event facilities module for events associated with business objects for Portal. This configuration file is called "pin_notify_cw" and is loaded into Infranet using the `load_pin_notify` utility delivered with Infranet.

When the event module receives an event, it extracts information from the event object and creates a new entry in the WebSphere Business Integration Adapter event table. An event in Infranet is actually an instance of an Infranet storable class, and each creation, modification, or deletion event is related to a specific Infranet storable class. For example, if a user modifies a particular contact related to a customer, Infranet generates an instance of the storable class `/event/customer/nameinfo`.

The event module uses its own event module configuration file to determine what event occurred, identify what part of the storable class (and related business object) was modified, and determine what type of action occurred. Using the configuration file `event_code.txt`, the event module examines the Infranet event and populates the WebSphere Business Integration Adapter event table with a record reflecting the event.

The event notification mechanism for the connector uses three tables created inside the Oracle database instance used by Infranet.

- XWORLDS_Events – Stores all pending events
- XWORLDS_Archive_Events – Archives events that the connector has processed
- XWORLDS_Current_Event_ID – Stores the last event ID number

The schema of the first table specifies the information that is recorded for each event sent by Infranet that the connector is interested in. This table layout is also used for the archive table.

Event detection and associated processing is done within an Infranet transaction. Infranet calls custom processes within a transaction and waits for the results of the processing. If the custom process returns an error, the transaction is aborted. This guarantees that the connector does not lose any events.

Note:

Known Issues - The event notification module verifies the USERID of any Portal Event sent to it defined in the pin_notify_cw file. If there is no PIN_FLD_USERID associated with the event sent to the module, it will error and cause problems saving the object online. These types of events need to be adjusted using FLists or the storable classes to include such an ID. Check for these errors in the log file defined in the crossworlds.cnf configuration file.

The event module checks for a USERID to prevent events sent into the application by the connector from being added to the event queue. This is referred to as Ping-Ponging.

The "/event/customer/billinfo" is an event type where such a problem exists.

Event retrieval

The connector checks for events by polling the XWORLDS_Events table that was set up in the Infranet database instance. The connector polls by using an SQL SELECT statement to extract entries from the XWORLDS_Events table. The number of events selected is specified by the PollQuantity property of the connector.

Polling is done by the `pollForEvents()` method in the connector. The connector polls the event table at the PollFrequency set in the WebSphere Business Integration Adapter connector properties. If a new row is detected in the table, the event data is retrieved, and the connector processes the event as follows:

1. The poll function creates an empty business object, sets the verb to Retrieve, and sets the keys using the event record. The business object is sent to the connector's business object handler.
2. The business object handler uses the event data to make a call to the Infranet Java API to retrieve the Portal Infranet storable object.
3. The business object handler converts the storable object to a WebSphere Business Integration Adapter application-specific business object, sets the verb to the action in the event record, and then sends the business object to the integration broker.

Once the business object is sent to the WebSphere business integration system, the event table entry is archived to the XWORLDS_Archive_Events table and deleted from the event table.

The time interval at which the poll method is called can be adjusted by changing the PollFrequency connector property. This property is set using the integration broker.

Connecting to the Infranet application

When connecting to the Portal Infranet connection manager using the API, the connector does the following:

1. Creates a new instance of a Portal Infranet context.
2. When an opcode is to be executed, the connector uses a context from the pool and adds it to the busy pool.
3. When the task is completed, the context is returned to the free pool.

The connect statement uses the values of connector application-specific properties that are defined in the repository.

The context instances in the pool are closed when the connector is terminated.

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 IBM CrossWorlds system are written in Java. Therefore, when data is transferred between most IBM CrossWorlds 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 connector properties,” on page 63

Chapter 2. Installing and configuring the connector

This chapter describes how to install and configure the connector and includes the following topics:

The connector component of the IBM WebSphere Business Integration Adapter for Portal Infranet has two components that need to be installed and configured:

- Connector. The connector is a Java `.jar` file. The connector implements the connector verb support and the event polling mechanism.
- WebSphere Business Integration Adapter event facilities module. The event facilities module is an executable that implements event notification. This module selects the Infranet events relevant for the integration broker and stores them in a database table.

This chapter describes how to install and configure the components of the connector and how to configure the Portal Infranet application to work with the connector.

Note: In this document backslashes (\) are used as the convention for directory paths, except in some example code files. For UNIX installations, substitute slashes (/) for backslashes (\). All file pathnames are relative to the directory where the WebSphere Business Integration Adapter product is installed on your system, unless otherwise noted.

Adapter for Portal Infranet environment

For hardware and software requirements for this adapter, see IBM WebSphere Adapters and IBM WebSphere Business Integration Adapters: Hardware and Software Requirements. Select your adapter from the list of WebSphere adapters.

Configuring the Infranet application

To set up the Infranet application for use by the connector, you must define a user account for the connector, and create the event and archive tables in the Oracle database used by Infranet.

Setting up an Infranet account

Using the Infranet Administrator, define a Customer Service Representative (CSR) User with all rights. This user is used by the connector and identifies the

connector. This user ID is set in the event module configuration file `crossworlds.cnf` and in the connector configuration parameters. The Custom Event Facilities module checks this value before inserting events, to prevent events sent into the application by the connector from being redistributed back to the connector. This scenario is also referred to as "ping-ponging."

Creating the event and archive tables in the database

Event and archive tables are used to queue events for pickup by the connector. The event notification mechanism for the connector requires that three event tables be created inside the Oracle database instance used by Infranet. These tables are:

- XWORLDS_Events – Event table used to store all pending Infranet events of interest to the connector.
- XWORLDS_Archive_Events – Archive table where events are written after the connector processes them.
- XWORLDS_Current_Event_ID – Table used to store the last event ID number.

  Note: This table *must* be initialized to 0.

The first two tables specify the information that will be recorded for each Infranet event that the connector is interested in. The archive table holds all events that have been processed by the connector.

To create the event and archive tables, load the file `EventTable.sql`if you are using an Oracle database. If you are using a DB2 database, then load the file `EventTable2.sql` in *`%ProductDir%`*`\connectors\Portal\dependencies\config_files`.

Description of event and archive table schema

The event table contains the following columns. This table layout is also used for the archive table.

Table 1. Event and archive table schema

Name	Type	Description
`Event_id`	integer	Unique key for the event. The key value is generated in the XWORLDS_Current_Event_ID table.
`Object_name`	char 80	Name of the application-specific business object.
`Object_verb`	char 80	Verb associated with the event.
`Object_key`	`VARCHAR`	Primary key for the object (POID).
`Event_time`	Date time	Time at which the event occurred.
`Archive_time`	Date time	Archive table only. Time at which the event was received by Portal Infranet.
`Event_status`	Integer	Status of the event: READY_FOR_POLL 0 SENT_TO_INTERCHANGE 1 UNSUBSCRIBED_EVENT 2 IN_PROGRESS 3 ERROR_PROCESSING_EVENT -1 ERROR_SENDING_EVENT_TO_INTERCHANGE -2
`Event_comment`	char 255	String used to provide extra information about the event. The user can define this comment in the event module configuration file.
`Event_priority`	Integer	Priority associated with the event. The lower the number, the higher the priority. The user can define this priority in the event module configuration file.

Installing the Portal Infranet adapter and other files

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

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

Configuring the adapter in an Oracle environment

If you are using Oracle as your database, follow these instructions to configure the connector:

1. Log into the UNIX system as `pin`. Create this account if necessary.
2. Copy the Infranet files `pcm.jar` and `pcmext.jar` into `$ProductDir/Connector/Portal/dependencies`. This file is located in the `infranet/jars` directory on the Infranet 6.7.0 server.
3. Copy the `.profile` file into the `pin` user's home directory, for example `/home/pin`. If necessary, modify the `.profile` file to reflect the environment variables set in your system. Make any changes using a text editor such as `vi`.

   When the environment variables are correct, load the environment variables into the system by typing the following command at the command prompt:

   ```
   source .profile
   ```
4. Place the `fm_crossworlds.so` file in the `$INFRANET/lib directory`. . This file contains the triggers for the events.

   Note that UNIX is case sensitive, so if files are not found, verify that all directory and file names have the proper case.
5. Make sure that the `$LIBRARY_PATH` variable contains the `$INFRANET/lib` path so that the system can recognize the connector`.so` files.
6. Copy the following files into the directory `$CW_PORTAL_PATH`.
   - `crossworlds.cnf`. This file contains configuration information for the event module.

     If necessary, edit this file for your system. An example of the content for this file is:

     ```
     db name = oracle1
     db string = NYNON
     db user = pin
     db password = pin
     crossworlds id = 0.0.0.1\service\admin_client 14088
     log level  = 3
     log file = D:\pinlog.log
     ```

     where:

`db name`	For Oracle, `db_name` is the host variable.
`db string`	Name of the database.
`db user`	Name of the user connecting to the Portal Infranet database.
`db password`	Password.
`crossworlds id`	POID representing the WebSphere Business Integration Adapter user in Portal Infranet.

log level	Number representing the log level: 0 : No trace 1 : Only Error 2 : Error and Warning 3 : Error, Warning, and Debug (all traces)
log file	Name of the log file.

Note: Provide values for db name and db string if your database is Oracle. If you have a default database on your local machine there is no need to provide values.

- event_code.txt. This file contains descriptions of Infranet events that the event module will use to generate entries in the WebSphere Business Integration Adapter event table.

7. Place the pin_notify_cw in the $INFRANET/sys/test directory. This file contains the names of the connector events. If any events need to be added or deleted, follow the standard format of the file. Note that /event encapsulates all subclasses, such as /event/customer, /event/status.
8. Stop and restart the Infranet application. Make sure that $INFRANET/bin is in the $PATH variable. Follow these steps:
   a. Stop Infranet with this command:
      ```
      stop_all
      ```
   b. Check that all Infranet processes are stopped by typing the following command. Note the process numbers (PID) of any running Infranet processes.
      ```
      ps -ef|grep portal
      ```
   c. Kill any running Infranet processes with this command:
      ```
      kill -9 <PID>
      ```
   d. Restart Infranet with the following command:
      ```
      start_all
      ```
9. In the $CM directory, edit the file pin.conf to add the following line in the fm_required section. Be sure to type the full directory path for $INFRANET.
   ```
   - cm fm_module $INFRANET/lib/fm_crossworlds.so fm_cw_pol_config -pin
   ```
10. Verify that Infranet is running by entering the command ps -ef|grep portal.
11. Change directory to $INFRANET/sys/test, open the pin.conf file, and check that the file has a line similar to the following.
    ```
    - nap cm ptr ip Infranet_cm_machine cm_port
    ```
    For example:
    ```
    - nap cm ptr ip roadrunner 11960
    ```
    where roadrunner is *Infranet_cm_machine* and *cm_port* is 11960.
    In addition to the above statement, the pin.conf file should include:
    ```
    - nap login_type 1
    - nap login_name root.0.0.0.1
    - nap login_pw password
    ```
    This identifies the login information for a connection to Infranet. If there is no pin.conf file in the directory, copy one into the directory.
12. To load configuration information into the Infranet application, enter the command:

```
load_pin_notify pin_notify_cw
```

The response should read `successful`. If another response is shown, check the `pin_notify_cw`. This file contains the opcodes that Infranet will call when specific events occur. Note that `pin_notify_cw` should be located in the same directory as the `load_pin_notify` executable.

13. In the `$INFRANET_VAR/cm` directory, check the log file and verify that there is a core in the `$CM` and then start up the Infranet Administrator.
14. To test the connector, enter or modify an account and check the event table `xworlds_events` for the proper event entry. Since this results in a dummy event, the event entry needs to be deleted once testing is complete.

To start the connector, see "Starting the connector" on page 21.

Configuring the connector

Adapters have two types of configuration properties: standard configuration properties and adapter-specific configuration properties. You must set the values of some of these properties before running the connector.

You configure connector properties from Connector Configurator (when WebSphere MQ Integrator Broker is the integration broker) or from Connector Configurator, which is accessed from System Manager (when ICS is the integration broker). For detailed configuration information, see Appendix B, "Using Connector Configurator," on page 87*Connector Development Guide for Java*.

Standard connector properties

Standard configuration properties provide information that all connectors use. See Appendix A, "Standard connector properties," on page 63 for documentation of these properties.

Important: Because this connector supports all integration brokers, configuration properties for all brokers are relevant to it.

Note: Because this connector is single-threaded, it cannot take advantage of the AgentConnections property.

Table 2 provides information specific to this connector about configuration properties in the appendix.

Table 2. Property Information Specific to This Connector

Property	Note
CharacterEncoding	This connector does not use this property.
Locale	Because the connector has been internationalized, you can change the value of this property. See release notes for the connector to determine currently supported locales.

Important: WebSphere MQ Integrator Broker does not support multiple locales. Ensure that every component of your installation (for example, all adapters, applications, and the integration broker itself), are set to the same locale.

Note: The Portal Infranet adapter supports the `DuplicateEventElimination` property. To enable the `DuplicateEventElimination` property, set the `DuplicateEventElimination` atribute to true.For more details about the `DuplicateEventElimination` property, see Appendix A, "Standard connector properties," on page 63.

Note: Because this connector supports all integration brokers, configuration properties for all brokers are relevant to it.

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 the connector.

Table 3 lists the connector-specific configuration properties for the connector. See the sections that follow for explanations of the properties.

Table 3. Connector-specific configuration properties

Name	Possible values	Default value	Required
Application password	*Password of user account*		Yes
ApplicationUserName	*Name of user account*		Yes
CommentFail	*Event table comment for failed events*	Fail	No
CommentSucceed	*Event table comment for successful events*	Succeed	No
DatabaseInfo	`jdbc:oracle:thin@CWENGTEST:1521:` portaldb	Server-host name, Portal database port number, and the name of the database.	Yes
DbName	*Oracle Listener name*		Yes
DbPassword	*Database password*		Yes
DbUser	*Database user name*	`pin`	Yes
DriverClass	Driver class name of the database		No
InfDatabase	*String*	`0.0.0.1`	Yes
InfHost	*Host machine name and port*	`//CWENGTEST1:11960`	Yes
InfLogFile	*Name of log file*	`InfConnection.txt`	Yes
InfranetConnections	*Number of Infranet connections the connector opens for connection pooling*	5	Yes
InfService	*String*	`service\admin_client 1`	Yes
InfType	`0` or `1`	`1`	Yes
InfVersion	*Version of the Infranet application*		Yes
PollQuantity	*Number of events to pick up*	`1`	No

Table 3. Connector-specific configuration properties (continued)

Name	Possible values	Default value	Required
UseDefaults	`true` or `false`	`false`	No

Application password
Password for WebSphere business integration system user account.

ApplicationUserName
Name of the user account for the connector.

CommentFail
Event table comment in case of a fail. Default value is `Fail`.

CommentSucceed
Event table comment in case of a success. Default value is `Succeed`.

DatabaseInfo
This property defines the URL that would be used by the JDBC driver to make a connection to the database.

Example:

```
jdbc:oracle:thin:@CWENGTEST1:1521:portaldb
```

Please refer to the JDBC documentation for specific formats of the URL

DbName
Oracle Listener name for the Infranet database.

DbPassword
Database password.

DriverClass
Driver class name of the database.

For Oracle set:

```
oracle.jdbc.driver.OracleDriver
```

If the DriverClass is not set, the adapter uses `oracle.jdbc.driver.OracleDriver` as the default value.

DbUser
Database user name, usually `pin`.

InfDatabase
String in Infranet form. The default value is `0.0.0.1`.

InfHost
Host machine name and port, for example, `//engtest2:11960`. The default value is `//CWENGTEST1:11960`.

InfLogFile
Name of file to use as default log. The default value is `InfConnection.txt`.

InfranetConnections
This property is used to define the number of connections the connector opens with the Infranet application for connection pooling. The connector maintains the assigned number of connections. When a business object process requires a connection, it allocates one from the pool. That connection is then removed from the available pool and added to the busy pool. When the business object process is complete, the connection is removed from the busy pool and added back to the available pool. Using connection pooling in this way increases efficiency in performance because the connection does not need to be opened and closed for each business process.

Note: If this property is not set, the connector will throw the following exception when the connector is started: `NumberFormatException`

InfService
String, usually `service\admin_client 1`. This is the default value.

InfType
Connection type. The only possible values are 0 or 1. The default value is 1.

InfVersion
Version of the Infranet application.

PollQuantity
Number of event picked up in a single poll. Default value is 1.

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

If the parameter is set to false, the connector checks only for valid values; the Create operation fails if valid values are not provided.

The default value is false.

Customizing the event mechanism for new business objects

If you create a new business object, you must determine what events will be generated for each action on the business object. When you have determined this, you must customize the event module configuration file so that the event module DLL can find events of that type. The name of the event module configuration file is `event_code.txt`, and it is located in `$INFRANET$\sys\cm`.

Infranet generates enough data for an event to enable the event module to identify what part of a storable class (in other words, what business object) has been invoked. When the event module gets an event, it gets an instance of a storable class containing information, such as the account, the user, and the calling program.

To differentiate an update from a delete or a create, the event module compares the original value and the updated one. To find out if a create or delete occurred, however, the event module must retrieve the storable class when the action is done at the root level, or it can look at the element ID for a child object. When a child is added, its element ID is positive and contains the position of this element in the array. If it is negative, it has been dropped.

Syntax of the event module configuration file

When modifying the event module configuration file, changes to the file must conform to the following syntax rules. This syntax must be strictly adhered to.

1. Comment lines begin with two dashes.
2. An event is described in one row. Parameters are delimited by the pipe character (|).
3. A row follows this syntax:

   `<event>|<Inf.action>|<array>|<key poid>|<constraints>|<BO.verb>|<priority>|<comment>`

   where:

event	Storable class name of the event
Infranet action	C (Create), U (Update), or D (Delete). This represents the action performed in Portal Infranet.
array	Portal Infranet code representing the array on which the action is done. The array is the Infranet element that must be retrieved from the event information. In Portal Infranet, each field has an associated number. For example, PIN_FLD_NAMEINFO is associated with the Infranet code 156. The list of fields and associated numbers can be found in the file `$INFRANET$\Include\pin_flds.h`.
poid	Portal Infranet code representing the field which is the key of the storable class created, updated, or deleted.
constraints	List of constraints necessary to determine exactly what happened. The constraint supports these keywords: • `exists` or `not_exists`: true or false specifying whether the object exists or not. • =, >, >=, <=, < : comparison operators for numerical types • `equal`, `nequal`, `contains`: comparison operators for string type. • `&` is used to separate constraints. The constraints are true only if all conditions separated by `&` are true. If you want to specify an `or`, use several lines. The first line in which the constraints are true is executed.
BO.verb	Name of the business object and verb corresponding to the action in Portal Infranet.
priority	Priority of the event
comment	Comment to insert in the event table

Example event module configuration file

The text below shows an example of an `event_code.txt` file. This example includes lines specifying events for the `account` storable class. Lines beginning with dashes are comments.

```
-- Account creation: PIN_FLD_STATUSES[0].PIN_FLD_STATUS[0] = 0 &
      PIN_FLD_SYS_DESCR = "Set Status (acct)": OK
/event/customer/status |U |144  |40  |40 exists&55;5 equal "Set Status
      (acct)"&144:0-145;3 = 0 |Portal_Account.Create |1 |Account Creation

-- Account Updated (status updated) : PIN_FLD_STATUSES[1].PIN_FLD_STATUS[0] =
      10100 or 10103 or 10102 & & PIN_FLD_SYS_DESCR = "Set Status (acct)" : OK
/event/customer/status |U |144  |40  |40 exists&55;5 equal "Set Status (acct)
      "&144:1-145;3 > 0 |Portal_Account.Update |1 |Status Updated

-- Account Updated (new contact added):OK
/event/customer/nameinfo |C |156 |40|40 exists|Portal_Account.Update|1|new contact
```

```
-- Account Updated (contact updated): OK
/event/customer/nameinfo |U |156 |40 |40 exists&17;5 equal "Customer Mngmt. Event
      Log" |Portal_Account.Update |2 |contact_update

-- Account Updated (contact deleted):OK
/event/customer/nameinfo |D |156  |40  |40 exists&67;5 nequal "Automatic Account
      Creation"|Portal_Account.Update |2 |contact_delete

-- Account Updated (billinfo updated) : two PIN_FLD_BILLINFO and
      PIN_FLD_BILLINFO[0].PIN_FLD_BILL_TYPE[0] <> 0 : OK
/event/customer/billinfo |U |126  |40 |40 exists & 126:0-127;3 > 0
      Portal_Account.Update |2 |billinfo_update
```

Defining event configuration file entries

When an event occurs, Infranet generates an flist representing the event. The event facilities module examines the flist to identify the verb and the action that occurred.

For example, assume an flist representing an event is:

```
0     PIN_FLD_POID            POID [0] 0.0.0.1 /event/customer/nameinfo -1 0
0     PIN_FLD_NAME             STR [0] "Customer Mngmt. Event Log"
0     PIN_FLD_USERID          POID [0] 0.0.0.1 /service/admin_client 2 1
0     PIN_FLD_SESSION_OBJ     POID [0] 0.0.0.1 /event/session 10366 0
0     PIN_FLD_ACCOUNT_OBJ     POID [0] 0.0.0.1 /account 9406 32
0     PIN_FLD_PROGRAM_NAME     STR [0] "Admin Manager"
0     PIN_FLD_START_T       TSTAMP [0] (942104217) 11/08/99 15:36:57
0     PIN_FLD_END_T         TSTAMP [0] (942104217) 11/08/99 15:36:57
0     PIN_FLD_SYS_DESCR        STR [0] "Set Name Info"
0     PIN_FLD_NAMEINFO       ARRAY [0] allocated 20, used 17
1     PIN_FLD_SALUTATION       STR [0] ""
1     PIN_FLD_LAST_NAME        STR [0] "Event Test1"
1     PIN_FLD_LAST_CANON       STR [0] "event test1"
1     PIN_FLD_FIRST_NAME       STR [0] "Event Test1"
1     PIN_FLD_FIRST_CANON      STR [0] "event test1"
1     PIN_FLD_MIDDLE_NAME      STR [0] ""
1     PIN_FLD_MIDDLE_CANON     STR [0] ""
1     PIN_FLD_TITLE            STR [0] "Event Test1  "
1     PIN_FLD_COMPANY          STR [0] "Event Test1"
1     PIN_FLD_ADDRESS          STR [0] "Event Test1"
1     PIN_FLD_CITY             STR [0] "Event Test1"
1     PIN_FLD_STATE            STR [0] "CA"
1     PIN_FLD_ZIP              STR [0] "00000"
1     PIN_FLD_COUNTRY          STR [0] "US"
1     PIN_FLD_EMAIL_ADDR       STR [0] ""
1     PIN_FLD_CONTACT_TYPE     STR [0] "Billing"
1     PIN_FLD_ELEMENT_ID      UINT [0] 1
0     PIN_FLD_NAMEINFO       ARRAY [1] allocated 20, used 19
1     PIN_FLD_SALUTATION       STR [0] ""
1     PIN_FLD_LAST_NAME        STR [0] "Event Test1"
1     PIN_FLD_LAST_CANON       STR [0] "event test1"
1     PIN_FLD_FIRST_NAME       STR [0] "Event Test1"
1     PIN_FLD_FIRST_CANON      STR [0] "event test1"
1     PIN_FLD_MIDDLE_NAME      STR [0] ""
1     PIN_FLD_MIDDLE_CANON     STR [0] NULL str ptr
1     PIN_FLD_TITLE            STR [0] "Event Test1"
1     PIN_FLD_COMPANY          STR [0] "Event Test1"
1     PIN_FLD_ADDRESS          STR [0] "Event Test1"
1     PIN_FLD_CITY             STR [0] "Event Test1"
1     PIN_FLD_STATE            STR [0] "CA"
1     PIN_FLD_ZIP              STR [0] "00000"
1     PIN_FLD_COUNTRY          STR [0] "US"
1     PIN_FLD_EMAIL_ADDR       STR [0] ""
1     PIN_FLD_CONTACT_TYPE     STR [0] "Billing"
1     PIN_FLD_ELEMENT_ID      UINT [0] 1
1     PIN_FLD_CANON_COUNTRY    STR [0] "US"
```

```
1     PIN_FLD_CANON_COMPANY    STR [0] "event test1"
0     PIN_FLD_ITEM_OBJ        POID [0] 0.0.0.1 /item 11454 0
0     PIN_FLD_CURRENCY        UINT [0] 840
```

When adding a line to the event module file, you must specify the number associated with the field instead of the field name. For example, the field PIN_FLD_NAMEINFO, as shown in bold in the preceding flist, is represented by the code 156.

For a field that is deeper than the first level of the flist, you must use a constraint to identify the field. With the exception of a constraint, each field is identified by a simple number, which is the number associated with the field.

Because a given event can be generated for various reasons, sometimes constraints are required to determine whether an event is the result of a particular business object. Therefore, you may need to add constraints such as `exists`.

The following syntax must be used in a constraint to identify a field.

```
[<array code:array element>-field;type]
```

For example, if the Infranet code 156 represents PIN_FLD_NAMEINFO, and the code 161 represents PIN_FLD_LAST_NAME, the representation of the field PIN_FLD_LAST_NAME contained in the array PIN_FLD_NAMEINFO of element ID 1 is represented in a constraint related to this field with 156:1-161;5 = PINFLDNAMEINFO[1].PIN_FLD_LASTNAME

The last number following the semicolon in a constraint represents the type of the object. The type indicates the type of data on which the comparison must be performed. For example, the number 5 indicates a string. For more information, see the list of associated types in `$INFRANET$\Include\pin_type.h`.

Adding events to the pin_notify_cw file

When you add new business objects, you must also add events to the `pin_notify_cw` file. This file contains the names of the connector events. To add events, follow the standard format of the file as seen in“Defining event configuration file entries” on page 18.

The CrossWorlds Event Notification Module (FmCw.dll) verifies the USERID of any Portal Event sent to it (defined in the pin_notify_cw file). If there is no PIN_FLD_USERID associated with the event sent to the module, it will error and cause problems saving the object online. These types of events need to be adjusted (f-lists or the storable classes) to include such an ID. Check for these errors in the log file defined in the crossworlds.cnf configuration file.

The Event Module checks for a USERID to prevent events sent into the application by the connector from being added to the event queue. This is referred to as Ping-Ponging. The ″/event/customer/billinfo″ is an example of an event type where such a problem exists.

Note: To identify all the events generated for an Infranet operation, call the event module for all Infranet events (using `/event` in the `load_pin_notify` configuration file). The event module will then be triggered for all events raised. Set the log level to 3 in the event module configuration file, and then in the `pinlog.log` file, you will have the event flist sent by Infranet for this operation.

Declaring Infranet custom attribute optional configurations

If the Infranet application has been customized, the corresponding Java classes must be generated. A tool is provided by Infranet to generate such classes. The connector must know these classes at runtime, so they must be declared in the CLASSPATH.

1. Create a directory, for example: c:\CustomAttributes.
2. Create an #include file with the custom fields and their values.
3. Launch the custom_fields.pl script on this file indicating com.portal.pcm.fields as the package name.
4. Compile the java files to create the class files:

   ```
   javac –classpath c:\program files\infranet\java\pcmext.jar *.java
   ```

5. Create a directory hierarchy under your current directory com\portal\pcm\fields.
6. Copy the .class generated by the compiler into this directory.
7. Add the current directory (for example: c:\CustomAttributes) to the CLASSPATH of the connector.
8. If needed, modify the file \bin\start_Portal.bat file.

Creating multiple connector instances

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

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

Create a new directory

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

```
ProductDir\connectors\connectorInstance
```

where connectorInstance uniquely identifies the connector instance.

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

```
ProductDir\repository\connectorInstance
```

Create business object definitions

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

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

   ```
   ProductDir\repository\initialConnectorInstance
   ```

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

Create a connector definition

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

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

Create a start-up script

To create a startup script:

1. Copy the initial connector's startup script and name it to include the name of the connector directory:

   dirname
2. Put this startup script in the connector directory you created in "Create a new directory" on page 20.
3. Create a startup script shortcut (Windows only).
4. Copy the initial connector's shortcut text and change the name of the initial connector (in the command line) to match the name of the new connector instance.

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

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

Starting the connector

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

ProductDir`\connectors\`*connName*

where *connName* identifies the connector.

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

The name of the startup script depends on the operating-system platform, as Table 4 shows.

Table 4. Startup scripts for a connector

Operating system	Startup script
UNIX-based systems	`connector_manager`
Windows	`start_`*`connName`*`.bat`

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

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

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

- On Windows systems, from the **Start** menu

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

    `start_connName connName brokerName [-cconfigFile ]`
  - On UNIX-based systems:

    `connector_manager -start connName brokerName [-cconfigFile ]`

  where *connName* is the name of the connector and *brokerName* identifies your integration broker, as follows:
  - For WebSphere InterChange Server, specify for *brokerName* the name of the ICS instance.
  - For WebSphere message brokers (WebSphere MQ Integrator, WebSphere MQ Integrator Broker, or WebSphere Business Integration Message Broker) or WebSphere Application Server, specify for *brokerName* a string that identifies the broker.

    Note: For a WebSphere message broker or WebSphere Application Server on a Windows system, you *must* include the `-c` option followed by the name of the connector configuration file. For ICS, the `-c` is optional.
- From Adapter Monitor, which is launched when you start System Manager running with the WebSphere Application Server or InterChange Server broker:

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

  You can load, activate, deactivate, pause, shutdown or delete a connector using this tool.
- On Windows systems, you can configure the connector to start as a Windows service. In this case, the connector starts when the Windows system boots (for an Auto service) or when you start the service through the Windows Services window (for a Manual service).

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

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

Stopping the connector

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

- If you started the connector from the command line, with its connector startup script:

- On Windows systems, invoking the startup script creates a separate "console" window for the connector. In this window, type "Q" and press Enter to stop the connector.
- When using InterChange Server on UNIX-based systems, connectors run in the background so they have no separate window. Instead, run the following command to stop the connector:

  ```
  connector_manager_connName -stop
  ```

  where *connName* is the name of the connector.

- From Adapter Monitor (WebSphere Business Integration Adapters product only), which is launched when you start System Manager:

  You can load, activate, deactivate, pause, shutdown or delete a connector using this tool.

- From System Monitor (WebSphere InterChange Server product only):

  You can load, activate, deactivate, pause, shutdown or delete a connector using this tool.

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

Chapter 3. Understanding business objects

This chapter describes how the connector processes business objects and provides guidelines for implementing business objects for Portal. The following topics are covered:

- "Portal Infranet application background"
- "Portal Infranet application-specific business object structure" on page 29
- "Guidelines for defining business objects" on page 32
- "Business object application-specific information" on page 32

Note: If you are not familiar with the Infranet application, designing and implementing business objects for Portal Infranet may be difficult. In this case, it is recommended that you work with an application expert. For information on Infranet concepts or programming elements, consult the Infranet documentation.

Portal Infranet application background

This section provides a brief overview of some basic elements of the Infranet application that affect the design and implementation of Portal Infranet application-specific business objects. Infranet defines four main programming elements that are used to define, extend, or access functionality in the system. In order to design business objects for Portal, you must be familiar with the following elements. They are briefly described in the sections that follow.

- Storable classes
- Storable objects
- Fields and field lists (flists)
- Opcodes

Storable classes and objects

In Infranet, storable classes contain fields that store information about a class. Standard storable classes include account, service, bills, invoices, and other classes that are predefined by Infranet. To extend Infranet functionality, you can create new storable classes or create subsets of existing classes.

Storable classes do not contain actual data; they are object specifications, much as a WebSphere Business Integration Adapter business object definition defines a business object structure but does not contain data. Storable classes include a number of fields, which can be simple fields (for example, an integer or string field), arrays, or substructures.

When a storable class has been instantiated and includes actual data values, it becomes a storable object. Each storable object is identified by a unique Portal Object ID, or POID. The POID contains the database number, the name of the storable class, the instance number of the storable object, and the object revision number.

The distinction between a storable class and storable object is illustrated in Figure 4.

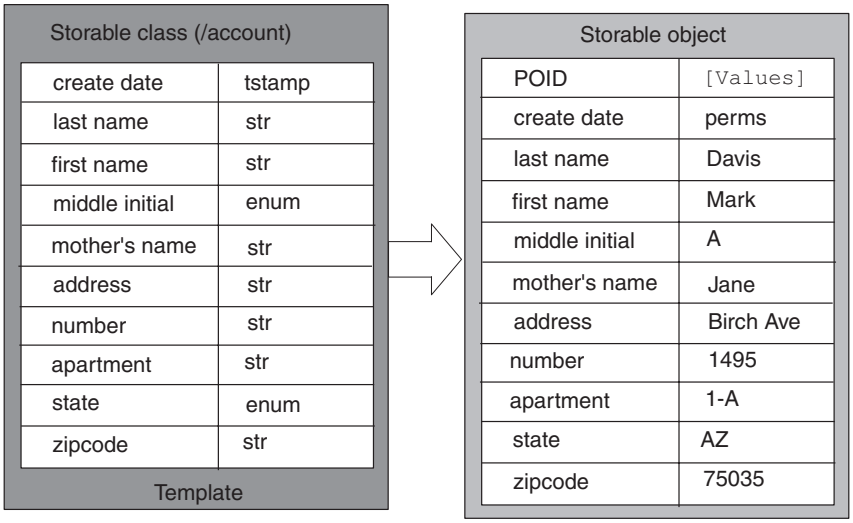


Figure 4. Infranet storable class and storable object

A storable class can define inherited and extended functionality for the class. For example, the `/account/email` storable class contains all the information in the `account` class with additional information that applies specifically to the `email` extended class. Therefore, the `/account/email` storable class becomes a subclass of `/account` as shown in Figure 5.

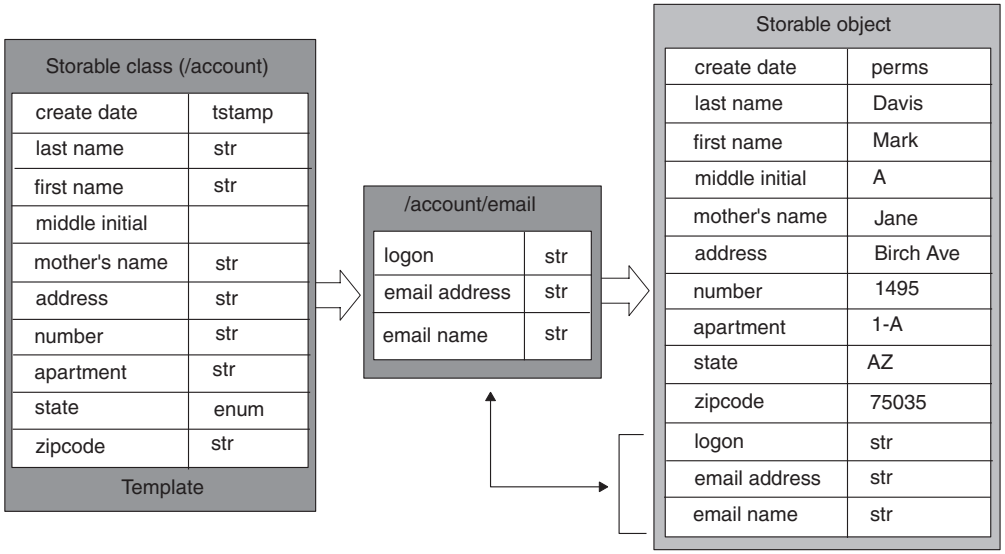


Figure 5. Extending the /account storable class

Storable objects are manipulated using Infranet application programs, scripts, and tools, or any custom programs and processes. Regardless of their type, all client programs operate on storable objects using the PCM API and programming libraries. Storable objects are manipulated by opcodes, which are routines containing lists of fields that operate on storable objects.

Fields and flists

Fields are the simplest data value in Infranet. Each field name in the system has a unique ID, name, type, and definition. Field names are shared and used in many different classes and opcode definitions.

There is a basic set of field types in the system that can be used to create new fields. Table 5 lists the field types. The first six types correspond to data types in programming languages such as C. The others hold more complex data and can point to C structures as their value. Arrays and substructs hold pointers to other lists of fields.

Table 5. Infranet field and data types

Field type	Data type
PIN_FLDT_INT	Signed integer
PIN_FLDT_UINT	Unsigned integer
PIN_FLDT_ENUM	Enumerated integer
PIN_FLDT_NUM	Floating point number
PIN_FLDT_TSTAMP	Time stamp
PIN_FLDT_STR	Character string
PIN_FLDT_BINSTR	Binary string
PIN_FLDT_BUF	Arbitrary-sized buffer of data
PIN_FLDT_POID	Portal Object ID
PIN_FLDT_ARRAY	Array
PIN_FLDT_SUBSTRUCT	Substructure

Field lists (flists) are fundamental data structures used in Infranet programming APIs. Flists are containers holding pairs of data fields and values, and in some cases, other flists. Flists can represent floating point calculations, buffers, or large pieces of data that do not fit in memory. Flists pass information between storable objects and the routines or programs that manipulate the storable objects.

A storable object (for example, in an /account storable class) makes up an flist (or part of an flist) that uses the storable class specification. The flist is a list of fields, each with its own attributes, permissions, and data values. Together, these fields define the functionality of the storable object, as shown in Figure 6.

Flists can contain multiple storable objects. The flist structure ensures that the information is passed from the application to the correct storable object.

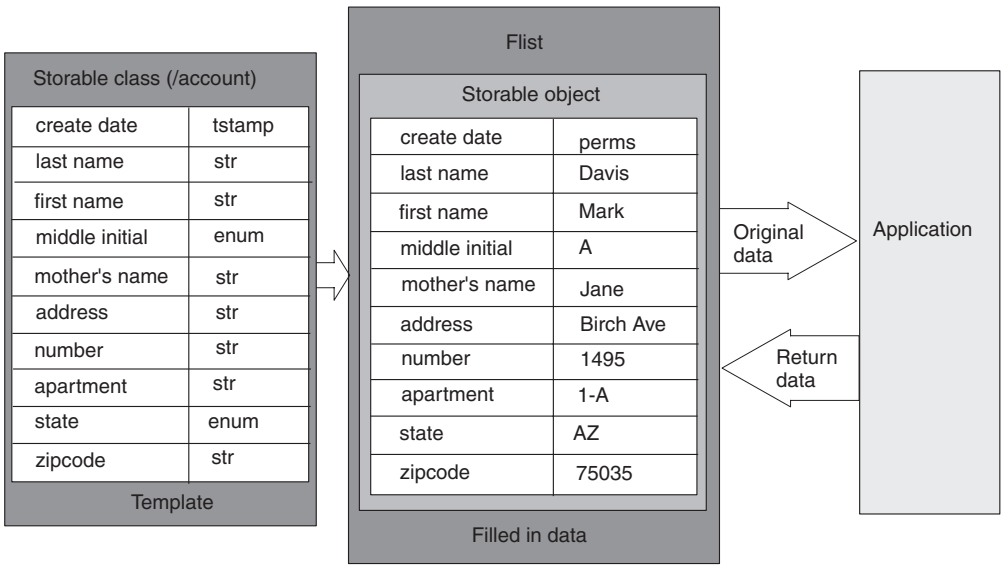


Figure 6. Storable object and flist

Opcodes

An application uses Infranet system opcodes to carry out operations on storable objects and the fields within them. There are several sets of opcodes, which are grouped into the following functional categories: Base, Customer facilities module (FM), Activity FM, Billing FM, Terminal FM, and Email FM.

Base operations on objects include creation, deletion, writing, reading, and searching. All other opcodes implement business-level (higher-level) semantics, such as logging activities, billing an account for the purchase of a product, checking credit card information, changing a name and address, verifying a password, or recording accounting data. These higher-level opcodes are implemented in facilities modules, where base opcodes are implemented directly by the Storage Manager (SM). The higher-level opcodes are translated by facilities module routines in the Communication Managers to varying numbers of base opcodes and then passed on to Storage Managers.

Every system opcode has an associated input and output flist. A client application determines what is an interesting event, calls the Infranet system with the appropriate opcode and corresponding flist, and handles the return flist and error buffer.

Meta-data-driven connector

The connector is a meta-data-driven connector. This means that the meta-data in the business object drives the behavior of the connector. Meta-data is data about the application that is stored in a business object and that assists the connector to interact with an 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.

The connector is meta-data driven using the Infranet `PIN_`*`FIELDNAME`* at the attribute level and the opcode value at the verb level. Because the connector is meta-data driven, it can handle new or modified business objects without requiring

modifications to the connector code. However, the connector makes assumptions about the following aspects of its business objects:

- Structure of its business objects
- Relationships between parent and child business objects
- Format of the application-specific information
- Database representation of a business object

Therefore, when you create or modify a business object for Portal Infranet, your modifications must conform to the rules the connector is designed to follow, or the connector will not be able to process new or modified business objects correctly. The following sections provide information on implementing business objects for Portal.

Portal Infranet application-specific business object structure

WebSphere Business Integration Adapter business objects are hierarchical: parent business objects can contain child business objects, which can in turn contain child business objects, and so on. A cardinality 1 container occurs when an attribute in a parent business object references a single child object. A cardinality n container object occurs when an attribute in the parent business object references an array of child business objects.

The connector supports both cardinality 1 and cardinality n relationships between business objects.

Corresponding Portal Infranet objects to WebSphere Business Integration Adapter business objects

Infranet has the following container types:

- Storable class
- Array
- Substruct

You must define a WebSphere Business Integration Adapter Portal Infranet application-specific business object so that it maps to the Infranet flist for the corresponding storable object with all required attributes and relationships. The relationship between an flist and a business object is a one-to-one relationship.

During processing, the connector compares a business object to the corresponding flist for the Infranet object, and it throws an exception if the structures do not match. It is possible to define a WebSphere Business Integration Adapter business object that is a subset of the flist structure, but the converse is not supported.

For each Infranet container type, an application-specific business object is created as needed. Typically, a storable class becomes a top-level business object. A container of type `substruct` may become a cardinality 1 child business object, and a container of type `array` may become a cardinality n child business object. However, if a subcontainer is not important and the parent opcode is sufficient to manipulate the children, a child business object is not needed.

Figure 7 shows how the structure of a WebSphere Business Integration Adapter business object and an Infranet flists might correspond. See the Portal Infranet documentation for information on Infranet flists.

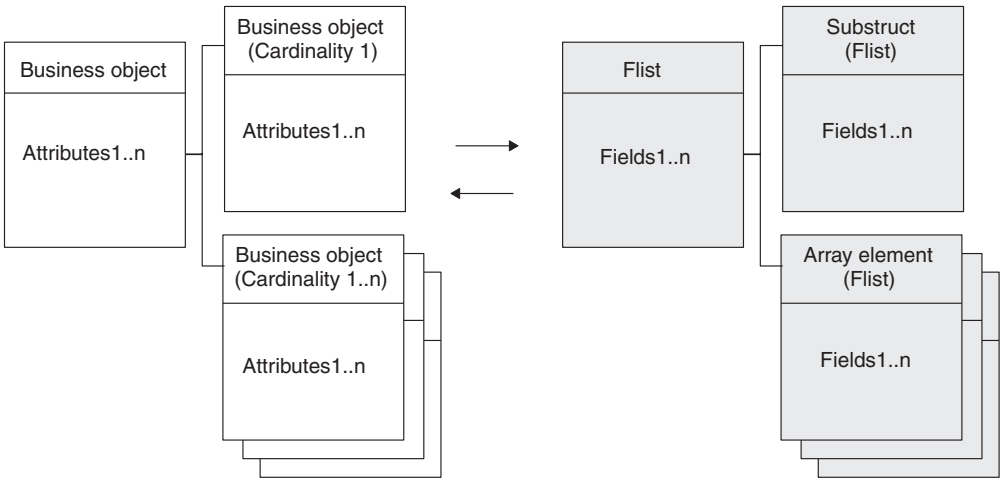


Figure 7. Structure business objects and flists

Figure 8 shows an example of a possible coordinating between the Infranet `/account` storable class and the Portal_Account hierarchical business object. The NameInfo array in the storable class becomes a cardinality n child business object in the top-level business object, and the Balances substruct becomes a cardinality 1 child business object.

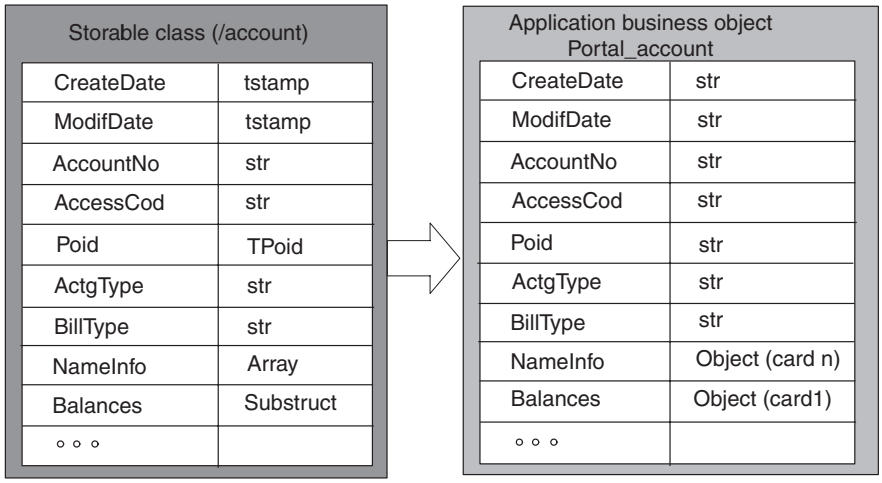


Figure 8. Coordinating of a storable class to a WebSphere Business Integration Adapter business object

Figure 9 shows a possible correspondence between the NameInfo array and the Portal_Contact child business object. The NameInfo array contains an array named Phones, which becomes a child business object whose parent is the Portal_Contact business object.

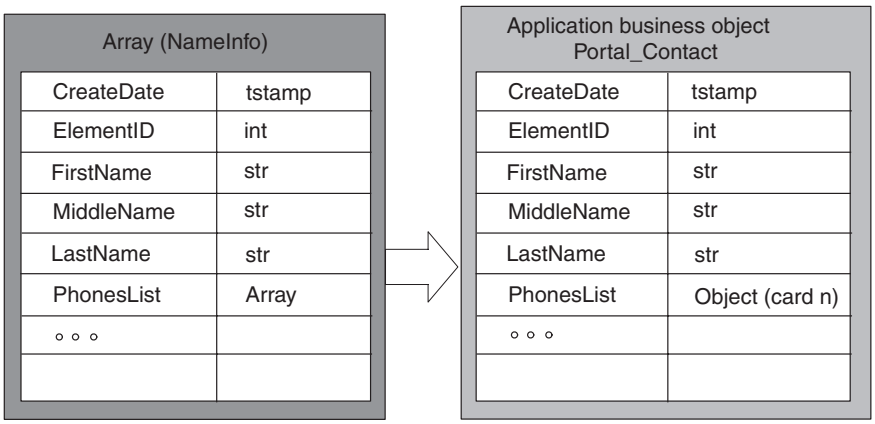


Figure 9. Corresponding of an flist array to a child business object

Note that specific attributes are needed for some flists and opcodes and not for others. In this case, an additional utility application-specific business object may be used as a verb parameter. This object does not correspond any persistent data; it describes only some mandatory fields for the flist. For more information on utility business objects, see "Connector utility business objects" on page 39.

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.

Key property

All business objects for Portal Infranet must have at least one key attribute. For each attribute that is a key, set the Key property to True.

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.

The key for a top-level business object is the storable object Portal Object ID (POID). A POID is a unique 64-bit identifier assigned to each Infranet database object when it is created. A POID is a unique key for an Infranet object.

A POID enables multiple instantiations of the same storable class. For example, every account object has a unique POID. A POID contains the following four components: a database number, an object type, a unique ID, and a revision number.

Key values for child business objects

An Infranet array is identified by its element ID, and the element ID is the unique key for the array. Because arrays typically correspond to child business objects, in Portal Infranet hierarchical business objects, a child business object key specifies the array element ID and the parent POID.

As a general rule, second-level child business objects simply need an attribute for the element ID. During a Create or Update operation, all the ElementId and POID values are filled in the business object.

Foreign key property

This property is used by the child business objects to relate to the parent business objects. During any verb processing, if the child is to be executed separately, the foreign key fields are populated from the parent business object. The name of which attribute from the parent business object is used is specified in the application-specific information on the foreign key attribute.

Required property

The connector does not use the Required property.

Max length property

Set the Max Length property to 255.

Default value property

The connector uses the default value of the attribute, if specified.

Guidelines for defining business objects

Use the following guidelines when defining a Portal Infranet application-specific business object:

- The POID attribute of an object must be the first attribute in the object definition.
- Any other ID attribute of an object should follow in the object definition.
- Any key or foreign key attributes should follow next.
- The remaining attributes, except for referenced and contained objects, should follow the key attributes, sorted logically.
- All referenced objects (those with cardinality 1) or contained objects (those with cardinality n) should follow next.
- The last attribute must be the ObjectEventId.

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. This meta-data is used with a business object's attribute properties and structure. When you create Portal Infranet application-specific business objects, 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, attribute, and verb application-specific information format for business objects for Portal.

Business object application-specific information

At the business object level, application-specific information describes the Infranet entity as follows:

- For a storable class, the business object application-specific information specifies the storable class name if the input flist for the Create or Update verb is similar to the structure of the business object.

- If the business object is a child object that corresponds to an flist array or substruct, the object application-specific information contains the Infranet field name.

For example, the Portal_Account business object specifies the `CN=/account` storable class for the object level application-specific information, and the Portal_BillInfo child business object specifies the `FN=PIN_FLD_BILLINFO` field for the object level application-specific information.

Attribute-level application-specific information

At the attribute level, application-specific information is used to prepare the flist structure that is used for executing a particular opcode. The application-specific information is a name-value pair list. This structure allows you to remove constraints previously imposed by the use of utility business objects, which are no longer required to define the structure of an flist for an opcode. The format of the application-specific information for an attribute is as follows:

```
FN=FIN_FLD_POID;Create=true;Update=false;Delete=true;
Retrieve=true;CreateT=;UpdateT=PIN_FL_NAMEINFO;
DeleteT=;RetrieveT=;O=false;CreateO=false;UpdateO=true;
DeleteO=false;RetrieveO=false;ParentAtt=;Alone=false
```

Converting old business objects to new business objects

The above format for attribute-level application-specific information is supported for backward compatibility in connector version 4.0.x. However, in future releases, backward compatibility will not be supported, and you must use PortalODA to convert old business object definitions to new business object definitions.

To use PortalODA to convert old business object definitions to new business object definitions, do the following:

1. Mark the foreign key fields in the child business objects which link the child business object to the parent business object. The application-specific information contains a tag called ParentAtt. Set the value for this tag to the name of the attribute from the parent business object which has to be used for the foreign key.
2. If necessary, mark attributes of type "object" with a specific verb tag. Refer to Table 3 for `CreatT`, `UpdateT`, `DeleteT`, and `RetrieveT`.

Table 3 on page 14 describes the format of the application-specific information for an attribute:

Table 6. Application-specific information for an attribute

Name	Description	Possible value	Default value
`FN`	Field name representing the field name in Infranet		
`Create`	Identifies whether the attribute is part of the flist for the Create verb	`true` or `false`	`false`
`Update`	Identifies whether the attribute is part of the flist for the Update verb	`true` or `false`	`false`
`Delete`	Identifies whether the attribute is part of the flist for the Delete verb	`true` or `false`	`false`
`Retrieve`	Identifies whether the attribute is part of the flist for the Retrieve verb	`true` or `false`	`false`

Table 6. Application-specific information for an attribute (continued)

Name	Description	Possible value	Default value
CreateT	Infranet field name that acts as a container for the attribute in the preparation of the flist for the Create verb		null
UpdateT	Infranet field name that acts as a container for the attribute in the preparation of the flist for the Update verb		null
DeleteT	Infranet field name that acts as a container for the attribute in the preparation of the flist for the Delete verb		null
RetrieveT	Infranet field name that acts as a container for the attribute in the preparation of the flist for the Retrieve verb		null
O	Identifies whether the attribute has to be updated in the response business object	true or false	false
CreateO	Identifies the Infranet field name that has the field representing the value for the attribute used to update the response business object for Create verb processing. For example, if the FN=PIN_FLD_POID and CreateO=PIN_FLD_NAMEINFO, then the connector looks for the field PIN_FLD_POID, and the value for this field is populated in the response business object for this attribute.		
UpdateO	Identifies the Infranet field name that has the field representing the value for the attribute used to update the response business object for Update verb processing.		
DeleteO	Identifies the Infranet field name that has the field representing the value for the attribute used to delete the response business object for Delete verb processing.		
RetrieveO	Identifies the Infranet field name that has the field representing the value for the attribute used to retrieve the response business object for Retrieve verb processing.		
ParentAtt	This field is used by a child business object to define the attribute in the parent business object that is used to populate the keys fields. These fields have to be marked as Foreign Key fields in the child business object.		NULL

Table 6. Application-specific information for an attribute (continued)

Name	Description	Possible value	Default value
`Alone`	This field is used by a child business object to represent that the child business object should be executed separately and not as part of the parent business object.	`true` or `false`	`false`

Figure 10 on page 36 shows the Portal_Account business object and illustrates application-specific information for the business objects and attributes.

Figure 10. Application-specific information for business object and attributes

Verb application-specific information format

The verb level application-specific information for business objects for Portal Infranet must specify a unique Infranet opcode for the action that the business object and verb will perform.

The opcode passes data between the Infranet application and its database and performs operations on storable objects. Each action on an Infranet object has a specific opcode.

Opcodes pass Infranet storable objects in the form of flists, which are lists of field name and value pairs. Each opcode has a specific input and output flist. The connector must convert a business object request into the input flist required by the opcode. When performing a business object request, the connector follows these basic steps:

1. Builds an input flist using the values in the business object instance and the information in the business object definition.
2. Executes the Portal Infranet opcode with the input flist as an argument.
3. The opcode returns an output flist, and the connector updates the business object with the information in the output flist.

The verb application-specific information and, in some cases, additional utility business objects enable the connector to generate the appropriate flists for each opcode. A utility business object enables the connector to build the correct input flist or to properly update the business object with the output flist. Utility business objects are provided as part of the business object definition for the top-level business object. The new definition of attribute-level application-specific information makes the utility business objects redundant. The feature of utility business objects is being supported in connector version 4.0.x for backward compatibility but would be removed from future releases.

Syntax of verb application-specific information

The required syntax of verb application-specific information is as follows:

```
<opcode>['#'<flag>][';'transaction
enabled][';'<input_flist_model>][';'<output_flist_model>]
```

where:

`opcode`	Any Infranet opcode without `PCM_OP_` at the beginning of the name, or `opcode` can be the keyword `ERROR`. The keyword `ERROR` indicates an Infranet constraint. The connector raises an error if the operation occurs. This might indicate that an operation cannot occur at a child level. For example a Portal_BillInfo business object cannot be deleted alone; it must be deleted with its parent object.
`flag`	An optional argument to an opcode. For information on opcode flags, see the Portal Infranet documentation.
transaction enabled	Some of the Opcodes in Portal Infranet maintain their own transaction due to the critical nature of Opcode functionality. This flag is used to provide that information to the Portal Infranet connector. The value “true” indicates that the Opcode maintains its own transaction. The default value used by the Portal Infranet connector for this property is “false.”

input_flist_model	Either the name of a utility business object for the input flist and, optionally, some parameters, or a keyword. The syntax is:<input_flist_model>[#<parameter>] The possible keywords are NotNull, NORMAL or OnlyPoid. These keywords are defined as follows: • NORMAL indicates that the current business object is the model; in other words, the connector can convert directly from the current business object to the input flist. • OnlyPoid indicates that the current opcode needs only the POID from the business object. The connector can then simplify its processing. • NotNull indicates that this object should not be deleted by setting the array to Null. See "Connector utility business objects" on page 39 for information on utility business objects.
output_flist_model	Either the name of a utility business object for the output flist and, optionally, some parameters, or a keyword. The syntax is:<output_flist_model>[#<parameter>] The possible keywords are NoRewrite and Flat. These keywords are defined as follows: • NoRewrite indicates that the output flist returned by the opcode must not be used to overwrite the business object. • Flat indicates that the connector should get the attributes for a child business object from the top-level business object. See "Connector utility business objects" on page 39 for information on utility business objects.

Rules of opcode application

One or more opcodes may exist at the storable class level for Create, Update, and Delete operations. However, the connector supports only one opcode for each verb. Therefore, for each verb, you must choose the opcode that is best suited to the business object and verb operation.

Different opcodes may be required at the parent and child levels. When an opcode exists for a child object, Portal Infranet advises using it rather than using the parent opcode. If a specific opcode is needed to update a subcomponent in Infranet, you must create a new application-specific business object for this child and specify the opcode in the verb application-specific information.

When the connector builds an input flist for a hierarchical business object, if a child business object verb has the same opcode as the parent, the connector puts the child in the same flist as the parent. Otherwise, the connector builds a separate flist for the child. Infranet uses levels in flists to specify arrays and substructures. When the connector starts building an input flist, it sets the level to zero. If the business object has children, the level is increased by an increment of one when processing a child object. If a business object needs a different opcode when executed alone rather than when executed as part of a hierarchy, then a different business object should be used with a similar structure.

For Create operations, parent opcodes are executed before child opcodes. For Delete operations, child opcodes are executed before parent opcodes. There is no mandatory order of execution for Update and Retrieve operations.

All storable classes can be retrieved using the opcode READ_OBJ and the POID of the root object.

Connector utility business objects

Each Infranet opcode requires a specific input flist and returns a specific output flist. In order for the connector to be meta-data driven, the business object must provide the connector with the fields that it needs to convert a business object instance and verb to the appropriate flist. Because opcode flists differ, you may not be able to build a single business object for Portal Infranet that provides all the information that the connector requires for every input and output flist. Instead, you may need to define special utility business object definitions that supplement the application-specific business object definition.

Note: The definition of attribute-level application-specific information for connector version 4.0.x does not require utility business objects. Connector version 4.0.x also supports backward compatibility for attribute-level application-specific information, but for future releases, you will need to use PortalODA to convert old business object definitions into new business object definitions. See "Converting old business objects to new business objects" on page 33for instructions.

Utility business object definitions do not become business object instances that are sent through the integration broker. The connector simply uses them to construct required input and output flists for specific opcodes. You must design and build utility business object definitions during the design of the application-specific business object, and you must define the connector to support all utility business objects as well as all application-specific business objects. For an example of a utility business object, see "Utility business object example: Create verb" on page 40.

To determine whether you need utility business objects, examine the Infranet storable class, and the input and output flists of the opcodes used to perform the verb operations.

Portal Infranet utility business objects use application-specific information that differs in format from that of business objects for Portal. This format is described in the next section.

Application-specific information for utility business objects

The attribute application-specific information in utility business objects specifies the fields to be added to an flist and contains the values to use for the flist fields. In utility business objects, you must define attribute application-specific information for simple attributes and for container attributes as follows.

- For a simple attribute, provide the field name for the flist and define the value for the field. A field can be extracted from the corresponding attribute in the business object instance, or it can be a default value provided in the application-specific information. The syntax for this description is:

  ```
  <flist_fieldname>[:[<bus_object_attributename>]:<default_value>]
  ```

- For array or structure attributes, the application-specific information contains a list of attributes in the application-specific business object that must be excluded from the flist. Attributes are separated by colon delimiters.

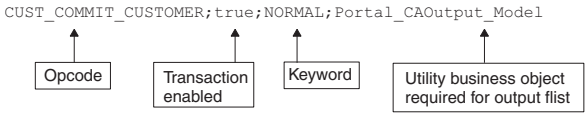
  ```
  [< bus_object_attributename>]([:< bus_object_attributename>])*
  ```

The verb description is not used for utility objects.

The following sections provide examples of application-specific information for Create, Update, Retrieve, and Delete verbs. The examples use the Portal_Account hierarchical business object to illustrate aspects of verb application-specific information.

Utility business object example: Create verb

As an example, consider the verb application-specific information for the Create verb in the Portal_Account top-level business object.



`CUST_COMMIT_CUSTOMER` is the opcode the connector uses to create a new customer account (an `/account` storable object). The opcode has its own transaction, so "Transaction Enabled" has been set to "true." The keyword `NORMAL` in the `input_flist_model` field indicates that the connector will correspond directly from the business object to the input flist. In other words, the business object provides all the fields required by the input flist, and the connector does not need supplemental information to create the input flist.

The `CUST_COMMIT_CUSTOMER` opcode for the Create Account operation returns an output flist containing the ID for the new customer in the `PIN_FLD_ACCOUNT_OBJ` field. The value of this ID must be returned to the WebSphere Business Integration Adapter system. To enable the connector to obtain the new ID, the business object designer created the Portal_C[reate]A[ccount]Output_Model utility business object definition. The `output_flist_model` field in the application-specific information specifies the Portal_CAOutput_Model as utility business object that the connector will use to read the output flist returned by the opcode.

The Portal_CAOutput_Model utility object contains one attribute, Poid, whose application-specific information tells the connector to extract the value of `PIN_FLD_ACCOUNT_OBJ` from the return flist to get the Portal Infranet object ID for the new customer. The connector inserts this value in the business object that it returns to the integration broker. The utility business object is shown in Figure 11.

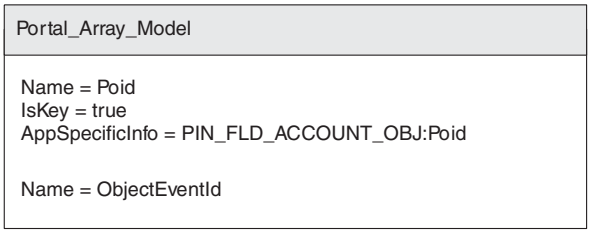


Figure 11. Portal_CAOutput_Model utility business object definition

Building flists for create operations: The Portal_Account business object is a hierarchical business object that looks like Figure 12.

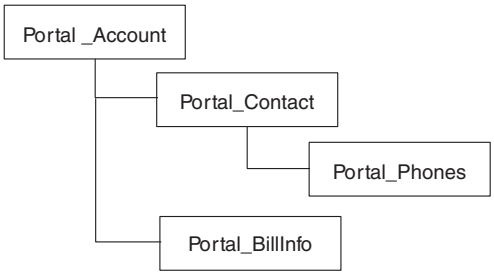


Figure 12. Diagram of the Portal_Account hierarchical business object

On a Create operation, the connector examines the verb application-specific information for the child business objects to determine if the opcode is the same as that used by the parent business object. For a Portal_Account business object, the opcode is the same for parent and child business objects, and the connector can build a single flist for the entire business object Create operation. Figure 13 illustrates the single opcode and flist that the connector uses to make the Create call to Infranet.

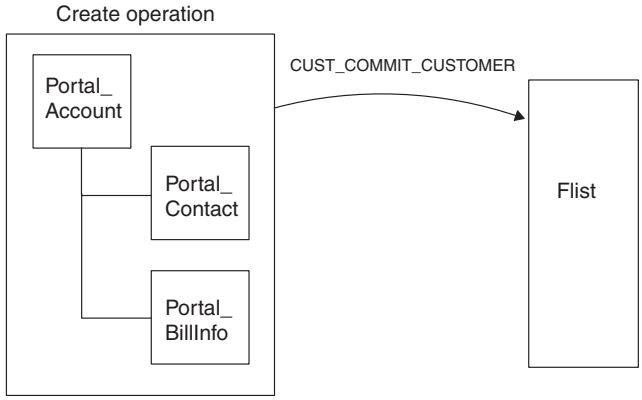


Figure 13. Flist for create operation for Portal_Account hierarchical business object

Note, however, that the connector must create the flist with arrays for the child business objects. Therefore, verb application-specific information must include fields that indicate the level in the flist that the array should occur. For example, the Create verb application-specific information for the Portal_Contact child business object is:

```
CUST_COMMIT_CUSTOMER
```

The Create verb application-specific information for the Portal_Phone child business object is:

```
CUST_COMMIT_CUSTOMER
```

Utility business object example: Update verb

This example shows the verb application-specific information for the Update verb in the Portal_Account top-level business object:

```
CUST_SET_STATUS;false;Portal_Array_Model#PIN_FLD_STATUSES;NoRewrite
```

`CUST_SET_STATUS` is the opcode required to update an account object. The opcode does not have its own transaction, so "Transaction Enabled" has been set to "false." For this verb operation, the connector cannot correspond directly from the business object instance to the flist because the Portal_Account business object does

not provide all the information required by the input flist. Because the connector needs additional information to create the flist, the `input_flist_model` field specifies the utility business object definition that the connector uses to construct the input flist. This utility object is named Portal_Array_Model.

The `output_flist_model` field contains the keyword `NoRewrite`, which indicates that the output flist returned by the opcode must not be used to overwrite the business object.

Portal_Array_Model utility business object: Portal_Array_Model is a hierarchical business object definition illustrated in Figure 14. It contains the information that the connector needs to build the input flist for the Update operation opcode. Specifically, the input flist requires an array that the Portal_Account business object definition does not contain. The Portal_Array_Model utility object enables the connector to create the array.

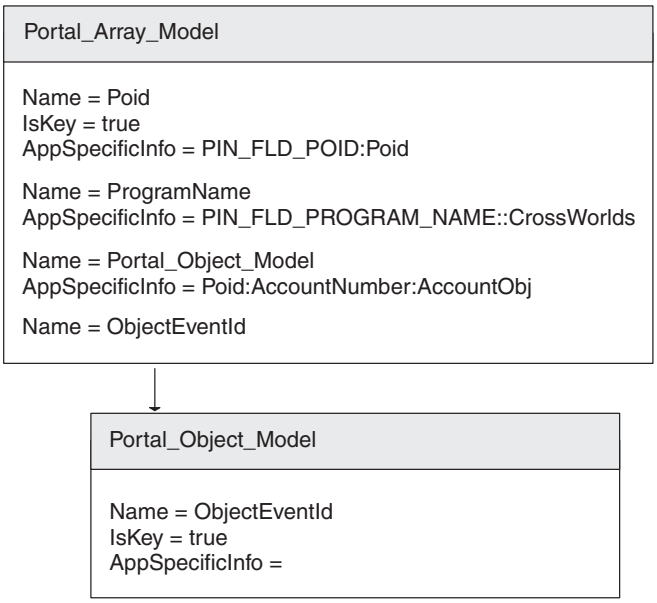


Figure 14. Portal_Array_Model utility business object definition

Recall that the Portal_Account Update verb application-specific information contained this text for the `input_flist_model` field.

```
Portal_Array_Model#PIN_FLD_STATUSES
```

This text identifies the utility business object used to create the flist and specifies the name of the array that the connector needs to put in the input flist. At runtime, the connector uses both the utility business object definition and the Portal_Account application-specific business object definition to build the input flist.

The connector builds the input flist as follows:

1. It begins constructing the flist with a field PIN_FLD_POID and gets the value of the POID from the business object instance.
2. It adds a field for PIN_FLD_PROGRAM_NAME to the flist. Because the Portal_Account business object definition does not contain this attribute, there is no value for this in the business object instance. Therefore, the value is defined in the application-specific information as the string `CrossWorlds`.

3. It adds an array named PIN_FLD_STATUSES to the flist. Because the flist for the opcode PCM_OP_CUST_SET_STATUS requires an array for PIN_FLD_STATUSES, the Portal_Array_Model must include a container attribute so that the connector will create an array in the flist. The connector names the array as indicated in the application-specific information for the Update verb.

   The connector uses the current business object, Portal_Account, as the model for the array. In other words, the connector will insert in the flist array the fields specified in the current business object. Because some of the fields may not be required, the application-specific information for the container attribute in the utility business object specifies which attributes to ignore. The container attribute Portal_Object_Model specifies these attributes:

   ```
   Poid:AccountNumber:AccountObj
   ```

   Therefore, to construct the array, the connector examines the business object definition for Portal_Account, ignores the POID, AccountNumber, and AccountObj attributes, and builds the array using only the remaining attributes in the business object definition, PIN_FLD_STATUS and PIN_FLD_STATUS_FLAGS.

As a result, the input flist for the Update verb looks like this:

```
PIN_FLD_POID              POID    <value from bus obj instance>
PIN_FLD_PROGRAM_NAME      STR     "CrossWorlds"
PIN_FLD_STATUSES          ARRAY
   PIN_FLD_STATUS         ENUM    <value from bus obj instance>
   PIN_FLD_STATUS_FLAGS     INT      <value from bus obj instance>
```

This flist contains the mandatory fields for the input flist for the PCM_OP_CUST_SET_STATUS opcode.

Child Business Object Processing: For an Update operation, the required Infranet opcodes are different to update an account storable object, update the customer contact information in the account storable object, and update the billing information in the account storable object.

Therefore, although the customer contact information and customer billing information are part of the same storable class, the connector must use different opcodes to update the Portal_Account top-level business object and the Portal_Contact and Portal_BillInfo child business objects. In addition, the connector must generate separate input flists for each opcode. Figure 15 shows the set of flists required to update the Portal_Account hierarchical business object.

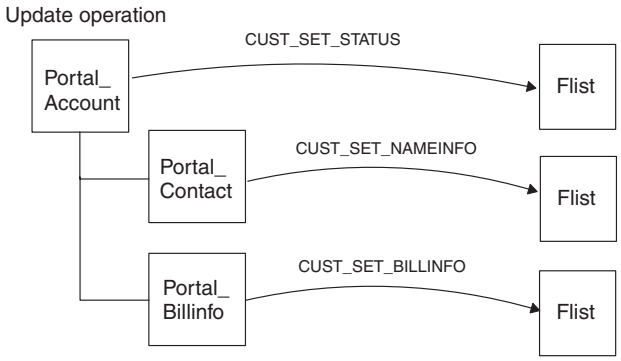


Figure 15. Flists for update operation for Portal_Account hierarchical business object

Utility business object example: Retrieve verb

This example is the Retrieve verb application-specific information in the Portal_Account top-level business object:

```
READ_OBJ;false;Only_Poid;Flat#Portal_BillInfo
```

The connector uses the READ_OBJ opcode to read a storable object from the database. This opcode does not have its transaction, so "Transaction Enabled" has been set to "false." In general, this opcode can be used for all Retrieve operations.

The input_flist_model field specifies that the current opcode needs only the POID from the business object. No other fields are needed for the input flist.

The opcode returns the POID of the object. All other fields in the object, including all array elements, are added to the return flist after the POID. The output_flist_model field contains the keyword Flat with the parameter Portal_BillInfo. This indicates that the information the connector needs to build the Portal_BillInfo child business object is contained in the flat flist rather than in an array.

Utility business object example: Delete verb

The final example is the Delete verb application-specific information in the Portal_Account top-level business object:

```
CUST_DELETE_ACCT;false;Only_Poid;NoRewrite
```

The connector uses the CUST_DELETE_ACCT opcode to delete the storable object from the database. This opcode does not have its own transaction, so "Transaction Enabled" has been set to "false." The input_flist_model field specifies that the current opcode needs only the POID from the business object. No other fields are needed for the input flist. The output_flist_model field contains the keyword NoRewrite, which indicates that the output flist returned by the opcode must not be used to overwrite the business object.

Note that Infranet is a logical delete application. For some objects, a delete operation is a change in status. For example, for the Portal_Service business object, the Delete verb application-specific information is:

```
CUST_SET_STATUS;Portal_DSInput_Model#PIN_FLD_STATUSES#NotNull
```

This text specifies the opcode for the logical Delete operation as CUST_SET_STATUS, and specifies that the input_flist_model is the utility object Portal_D[elete]S[ervice]Input_Model, which defines an flist with a PIN_FLD_STATUSES array that is not set to Null.

Context-driven verb behavior

Because the connector has to be able to handle both a hierarchical business object and a single child business object (for example, a Portal_Contact business object can be sent without the parent), certain meta-data-driven decisions depend on the context of the business object and verb. Depending on the verb, you must specify one unique opcode for the whole business object hierarchy or one opcode for each business object. For this reason you have to specify the opcode used by each level in each business object.

For a verb, the global opcode (the parent opcode) is applied if both the opcode for the child level and the parent level are similar. Otherwise, the parent opcode on the parent is applied first, followed by the child opcode for each child.

For example, when you need to create a contact, if the Portal_Contact business object is sent as an individual business object, use the opcode CUST_SET_NAMEINFO for the Create verb application-specific information. However, if the contact will be created with the account business object, use the parent opcode CUST_COMMIT_CUSTOMER. This opcode must be specified in the application-specific information. To support the above functionality, two copies of the Portal_Contact business object must be created and used for two different opcodes. One business object would have the verb ASI set to CUST_SET_NAMEINFO and the other business object would have the verb ASI set to CUST_COMMIT_CUSTOMER.

A complete sample Portal Infranet business object definition

The following structure describes the properties and application specific information for Sample Account business object in Portal Infranet adapter.

```
[BusinessObjectDefinition]
Name = Portal_Account
Version = 1.0.0
AppSpecificInfo = CN=/account
[Attribute]
Name = Poid
Type = String
Cardinality = 1
MaxLength = 255
IsKey = true
IsForeignKey = false
IsRequired = true
AppSpecificInfo = eu
IsRequiredServerBound = false
[End]
[Attribute]
Name = AccountNumber
Type = String
Cardinality = 1
MaxLength = 255
IsKey = false
IsForeignKey = false
IsRequired = false
AppSpecificInfo =
   UpdateO=;FN=PIN_FLD_ACCOUNT_NO;Create=true;O=true;DeleteT=;
   Update=falTse;RetrieveT=;Alone=false;CreateT=;Retrieve=false;
   DeleteO=;ParentAtt=;UpdateT=;RetrieveO=Main;Delete=false;CreateO=
IsRequiredServerBound = false
[End]

[Attribute]
Name = AccountObj
Type = String
Cardinality = 1
MaxLength = 255
IsKey = false
IsForeignKey = false
IsRequired = false
AppSpecificInfo =
   UpdateO=;FN=PIN_FLD_ACCOUNT_OBJ;Create=true;O=true;DeleteT=;Update=false;
   RetrieveT=;Alone=false;CreateT=;Retrieve=false;DeleteO=;ParentAtt=;
   UpdateT=;RetrieveO=Main;Delete=false;CreateO=
IsRequiredServerBound = false
[End]

[Attribute]
Name = Status
Type = String
Cardinality = 1
MaxLength = 255
IsKey = false
```

```
IsForeignKey = false
IsRequired = false
AppSpecificInfo =
   UpdateO=;FN=PIN_FLD_STATUS;Create=true;O=true;DeleteT=;Update=true;
   RetrieveT=;Alone=false;CreateT=;DeleteO=;Retrieve=false;ParentAtt=;
   RetrieveO=Main;UpdateT=PIN_FLD_STATUSES;CreateO=;Delete=false
IsRequiredServerBound = false
[End]

[Attribute]
Name = StatusReason
Type = String
Cardinality = 1
MaxLength = 255
IsKey = false
IsForeignKey = false
IsRequired = false
AppSpecificInfo =
   UpdateO=;FN=PIN_FLD_STATUS_FLAGS;Create=true;O=true;DeleteT=;
   Update=true;RetrieveT=;Alone=false;CreateT=;DeleteO=;Retrieve=false;
   ParentAtt=;RetrieveO=Main;UpdateT=PIN_FLD_STATUSES;CreateO=;Delete=false
IsRequiredServerBound = false
[End]

[Attribute]
Name = Portal_Contact
Type = Portal_Contact
ContainedObjectVersion = 1.0.0
Relationship = Containment
Cardinality = n
MaxLength = 0
IsKey = false
IsForeignKey = false
IsRequired = false
AppSpecificInfo =
   UpdateO=Main;FN=PIN_FLD_NAMEINFO;Create=true;O=true;Update=true;
   Alone=false;Retrieve=false;DeleteO=Main;ParentAtt=;RetrieveO=Main;
   CreateO=Main;Delete=false
IsRequiredServerBound = false
[End]

[Attribute]
Name = PlaceHolder
Type = String
Cardinality = 1
MaxLength = 255
IsKey = false
IsForeignKey = false
IsRequired = false
IsRequiredServerBound = false
[End]

[Attribute]
Name = Portal_BillInfo
Type = Portal_BillInfo
ContainedObjectVersion = 1.0.0
Relationship = Containment
Cardinality = n
MaxLength = 0
IsKey = false
IsForeignKey = false
IsRequired = false
AppSpecificInfo =
   UpdateO=Main;FN=PIN_FLD_BILLINFO;Create=true;O=true;Update=true;
   Alone=false;Retrieve=false;DeleteO=Main;ParentAtt=;RetrieveO=Main;
   CreateO=Main;Delete=false
IsRequiredServerBound = false
```

```
[End]

[Attribute]
Name = ProgramName
Type = String
Cardinality = 1
MaxLength = 1
IsKey = false
IsForeignKey = false
IsRequired = false
AppSpecificInfo =
   UpdateO=Main;FN=PIN_FLD_PROGRAM_NAME;Create=false;O=true;DeleteT=;
   Update=true;RetrieveT=;Alone=false;CreateT=;DeleteO=Main;Retrieve=false;
   ParentAtt=;PAttName=;RetrieveO=Main;UpdateT=;CreateO=;Delete=true
DefaultValue = CrossWorlds
IsRequiredServerBound = false
[End]

[Attribute]
Name = ObjectEventId
Type = String
Cardinality = 1
MaxLength = 255
IsKey = false
IsForeignKey = false
IsRequired = false
IsRequiredServerBound = false
[End]

[Verb]
Name = Create
AppSpecificInfo =
   IFM=;OpCode=CUST_COMMIT_CUSTOMER;OFP=;TFlag=true;
   IFP=;IF=NORMAL;Flag=0;OF=NORMAL
[End]

[Verb]
Name = Delete
AppSpecificInfo =
   IFM=;OpCode=CUST_DELETE_ACCT;OFP=;TFlag=false;IFP=;
   IF=NORMAL;Flag=0;OF=NORMAL
[End]

[Verb]
Name = Retrieve
AppSpecificInfo =
   IFM=;OpCode=READ_OBJ;OFP=Portal_BillInfo;TFlag=false;IFP=;
   IF=NORMAL;Flag=0;OF=NORMAL
[End]

[Verb]
Name = Update
AppSpecificInfo =
   IFM=;OpCode=CUST_SET_STATUS;OFP=;TFlag=false;
   IFP=;IF=NORMAL;Flag=0;OF=NORMAL
[End]

[End]
```

Chapter 4. Generating business object definitions using PortalODA

This chapter describes PortalODA, an object discovery agent (ODA), which generates business object definitions for the connector. The PortalODA uses the Portal Infranet APIs to get the information about the Portal Infranet storable classes. It then uses this information to build new business object definitions. The PortalODA also enables the conversion of existing business object definitions to those which are supported by the connector.

The following topics are covered:

Installation and usage

This section discusses the following:

Installing PortalODA

To install PortalODA, use the WebSphere Business Integration Adapter (WBIA) Installer. 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\Portal\PortalODA.jar`
- `ODA\messages\PortalODAAgent.txt`
- `ODA\Portal\start_PortalODA.bat` (Windows only)
- `ODA/Portal/start_PortalODA.sh` (UNIX only)
- `bin\CWODAEnv.bat` (Windows only)
- `bin/CWODAEnv.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 WBIA product pathnames are relative to the directory where the product is installed on your system.

Before using PortalODA

Before you can run PortalODA, you must copy the required Portal Infranet application's.jar files to the *%ProductDir%*/connectors/Portal/dependencies directory. The following files must be copied to this directory:

```
pcm.jar
pcmext.jar
```

The above files are present in the %INFRANET%\jars folder.

After installing the PortalODA, you must do the following to generate or convert business objects:

1. Start the ODA.
2. Start 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.

Starting PortalODA

You can start PortalODA using one of the following scripts:

UNIX:

```
start_PortalODA.sh
```

Windows:

```
start_PortalODA.bat
```

You configure and run PortalODA 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 PortalODA.

Running PortalODA on multiple machines

You can run multiple instances of the ODA, either on the local host or a remote host in the network. Each instance has to run on a unique port if multiple instances are being run on the same machine.

Figure 16 on page 52 illustrates the window in Business Object Designer from which you select the ODA to run.

Changing the error and message filename

The error and trace message file (PortalODAAgent.txt) is located in \ODA\messages\, which is under the product directory. This file uses the following naming convention:

```
AgentNameAgent.txt
```

If you change an ODA's name in the AGENTNAME variable of a script or batch file, use this convention to change the name of its associated error and trace message file.

If you create multiple instances of the script or batch file and provide a unique name for each represented ODA, create a copy of the error and trace message file

for each of these. Name each file according to this convention. For example, if the `AGENTNAME` variable specifies `PortalODA1`, name the associated message file: `PortalODA1Agent.txt`.

During the configuration process, you specify:

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

Table 7 describes the tracing level values.

Table 7. 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 PortalODA spawned • Traces the business object definition dump

For information on where you configure these values, see "Configure initialization properties" on page 52.

Using PortalODA in business object designer

This section describes how to use PortalODA in Business Object Designer to convert the existing business definitions to new ones and to generate new business object definitions. This is done by getting information directly from Portal Infranet. For information on starting Business Object Designer, see the *Business Object Development Guide*.

After you start an ODA, you must start Business Object Designer to configure and run it. There are six steps in Business Object Designer to generate or convert 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 16 on page 52 illustrates this window.

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

1. "Select the ODA" on page 52
2. "Configure initialization properties" on page 52
3. "Generating definitions" on page 56 and, optionally, "Providing additional information" on page 57
4. "Saving definitions" on page 58

Select the ODA

Figure 16 on page 52 illustrates the first dialog box in Business Object Designer's six-step wizard.

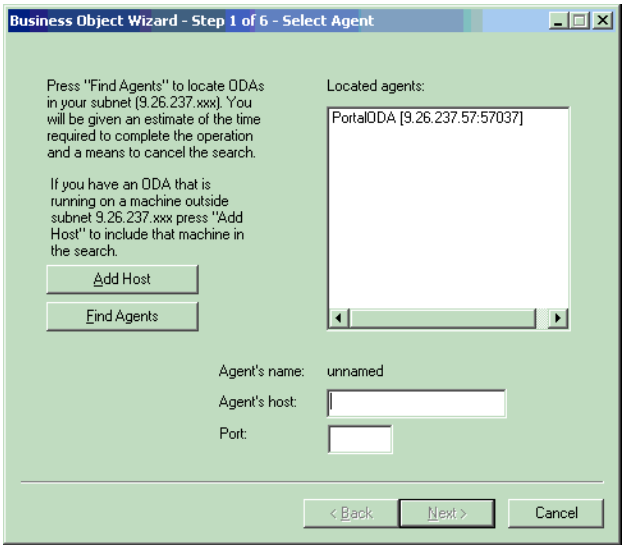


Figure 16. Selecting the ODA

To select the ODA:

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

   You can also find the agent using the Host name and the Port number.

   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 PortalODA, it prompts you to enter a set of initialization properties as shown in Figure 17 on page 53. You can save these properties in a named profile so that you do not need to re-enter them each time you use PortalODA. For information on specifying an ODA profile, see the *Business Object Development Guide*.

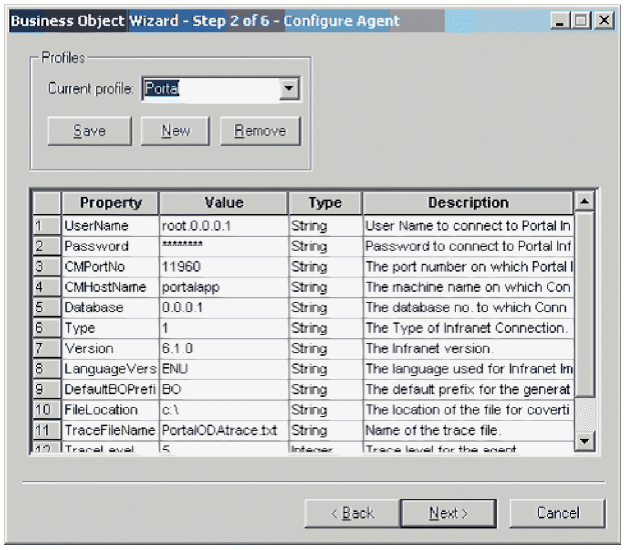


Figure 17. Configuring agent initialization properties

Configure the PortalODA properties described inTable 8.

Important: All of the PortalODA properties inTable 8 are required to be entered.

Table 8. PortalODA properties

Row number	Property name	Property type	Description
1	`UserName`	String	Portal Infranet application login name
2	`Password`	String	Portal Infranet application password
3	`CMPortNo`	String	The port number on which the connection manager is running
4	CMHostName	String	The name or IP address of the machine on which the connection manager is running
5	Database	String	The database number to which the connection manager is connected
6	Type	String	The Portal Infranet connection type: 1 is for validating UserName and Password, and 0 is for no validation
7	Version	String	Version of Portal Infranet
8	`LanguageVersion`	String	Example: ENU for English
9	`DefaultBOPrefix`	String	Example: Portal_BO
10	FileLocation	String	The absolute path containing the files with previous versions of business object definitions. For example, in Windows, if the path is `C:\PortalBos`, you must enter the value `C:\\Portal\\`In UNIX, if the path is `/home/PortalBOs`, you must enter the value `/home/PortalBos/`
11	TraceFileName	String	Name of the trace file
12	`TraceLevel`	Integer	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 57
13	MessageFile	String	Path to the message file

Expand nodes and select repository files, and storable classes

After you configure all initialization properties for PortalODA, the following screen is displayed by Business Object Designer.

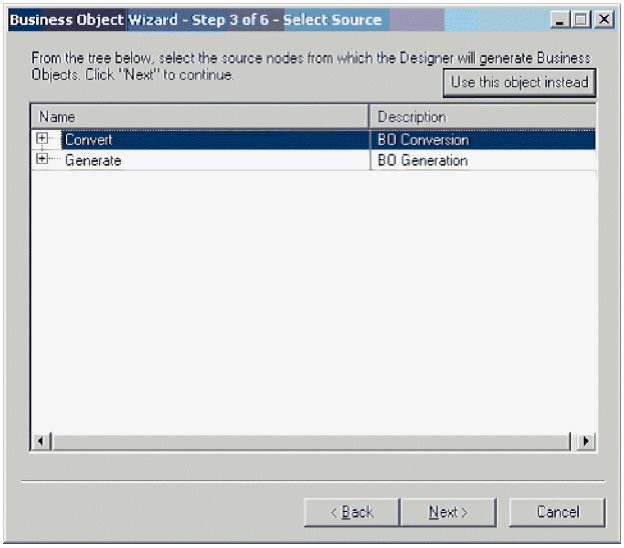


Figure 18. Tree giving two options for BO conversion and BO generation

This screen has two expandable options, Convert and Generate. If you need to convert the old business object definitions into new ones, expand Convert. This displays the repository files containing the business object definitions that need to be converted.

Converting old business object definitions

The old business object definitions have application-specific information as comma delimited values while the new business object definitions have application-specific information as name-value pairs which are comma delimited. Also, the old business object definitions use meta business objects to transform the structure of a business object for a particular opcode while in the new business object definitions, this feature is replaced with the name-value pair of application-specific information at the attribute level of the business object.

Select the files to be converted, then click Next.

Note: When you select a file, all of the business object definitions in that file are converted. There is no method for selecting a subset of business object definitions to convert. However, if you want to convert only a subset of business object definitions, you can create a new file with a subset of business object definitions, then convert the new file.

Generating new business objects

If you need to generate new business object definitions by getting information from Portal Infranet, expand Generate. This gets all of the storable class names from Portal Infranet and displays a tree.

The storable class names which are presented as nodes in the tree are expandable (see Figure 19 on page 55). The generated business objects have some properties which have to be set individually before the business object can be used by the

connector. The key fields for any business object have to be marked as key fields in the WebSphere business integration system business object. Depending on the opcode being used for the different verbs, the attribute-level application-specific information has to be set. For example, if an attribute is part of the Create verb opcode, the value for the property "Create" should be set to the name of the parent field. Refer to "Attribute-level application-specific information" on page 33 for details of various properties in application-specific information of an attribute.

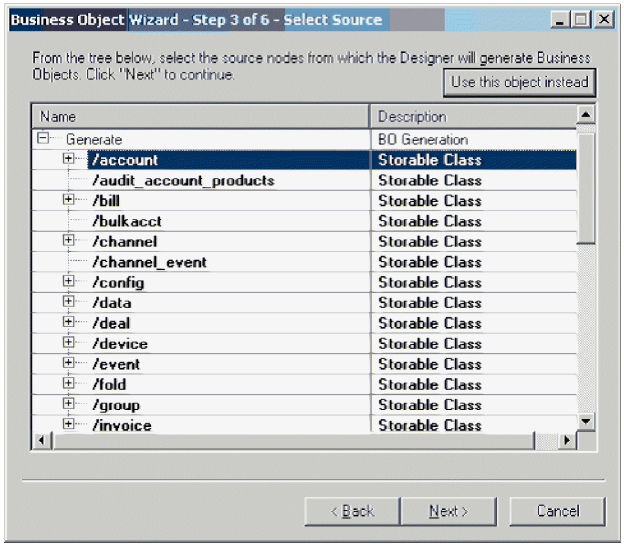


Figure 19. Screen showing the storable classes

This screen allows you to choose a storable class from the list to generate. The "+" sign before the class name means that the class has child objects. Multiple classes can be selected for generation.

Note: When you select a class to be generated which has child objects, the child objects are not selected by default. You must explicitly select the child objects if you want to generate those as well. You can do this by holding the Shift key while selecting the child object.

Confirming the selection of the repository files and storable classes

After you identify all the repository files or storable classes to be associated with the generated business object definition, Business Object Designer displays the following confirmation screen (see Figure 20 on page 56).

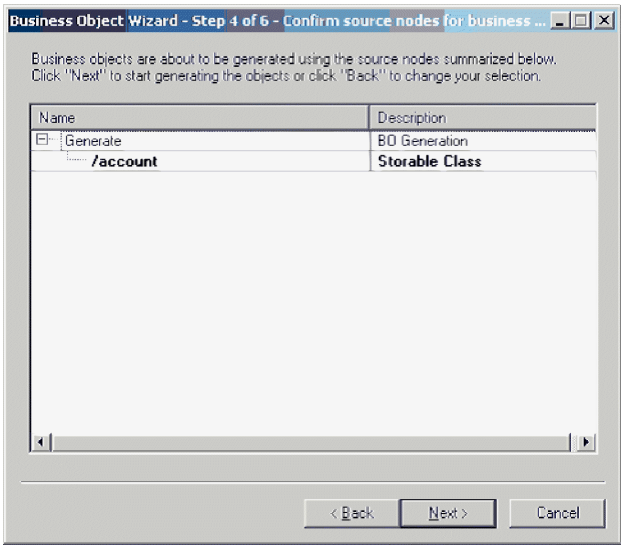


Figure 20. Confirming your selection

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.

Figure 21 on page 56 illustrates this dialog box.

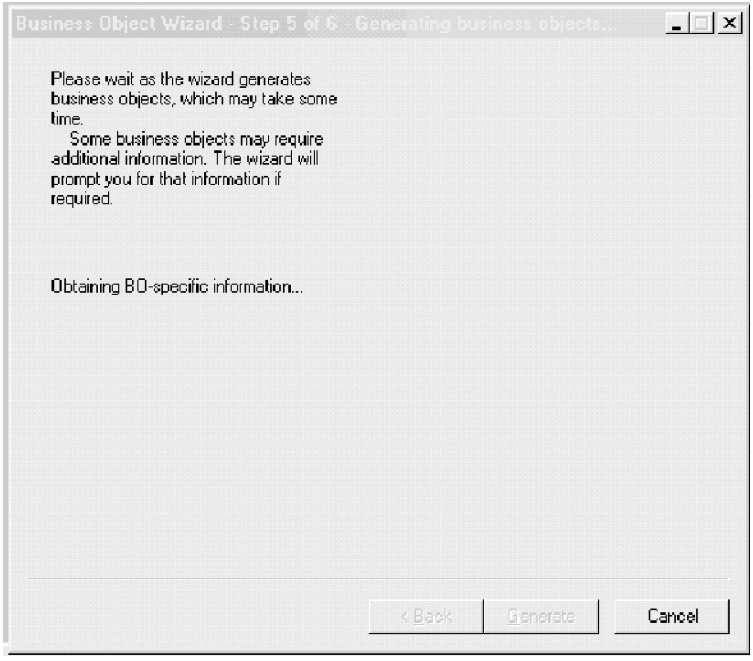


Figure 21. Generating definitions

Providing additional information

If the PortalODA needs additional information, Business Object Designer displays the BO Properties window, which prompts you for the information. This is done only in the case of business object generation. Figure 22 on page 57 illustrates this window.

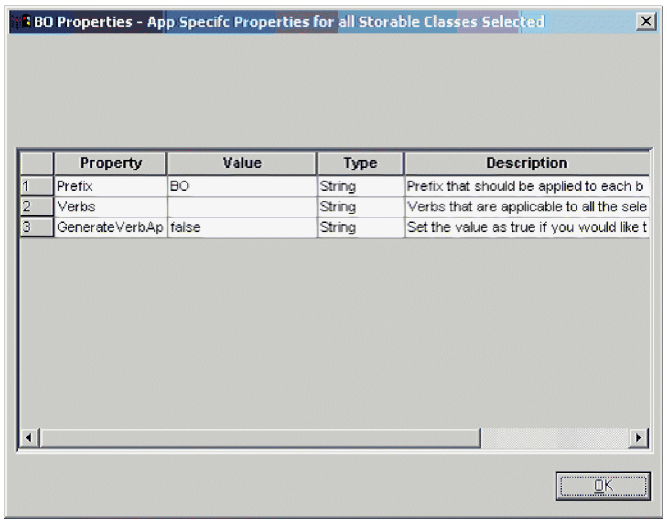


Figure 22. Application-specific properties for storable classes

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 17 on page 53), 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.

  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.
- *GenerateVerbApp*—A flag which allows you to edit the application-specific information at the verb level.

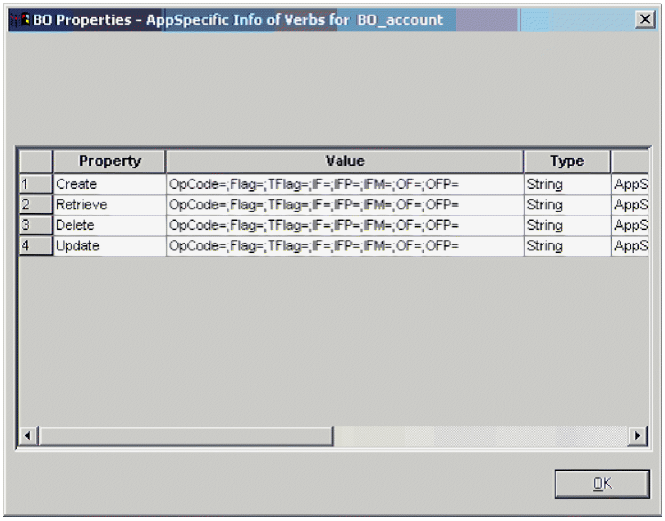


Figure 23. Application-specific information for verbs

The format for the verb-level application-specific information is:

`OpCode=;Flag=;TFlag=;IF=;IFP=;IFM=;OF=OFP=` describes

Table 9 describes each name in the verb-level application-specific information.

Table 9. Application-specific information for verbs

Name	Description
`Opcode`	The name of the opcode which should be executed for this verb
`Flag`	The flag value which should be used with the Opcode
`TFlag`	TFlag is either true or false depending on whether the Opcode maintains its own transaction or not.
`IF`	Input Flist (IF) is the name of the business object that is used to prepare an input flist for the opcode
`IFP`	Input Flist Parameter (IFP) is the name of the optional parameter that can be used to prepare the input flist.
`IFM`	Input Flist Mode (IFM) is the value that defines the kind of flist translation that is done
`OF`	Output Flist (OF) is the parameter that governs how the return flist of the opcode execution should be converted to a business object
`OFP`	Output Flist Mode (OFM) is the value that defines the kind of business object update that is done from the output flist of the opcode

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*.

Figure 24 on page 59 illustrates this dialog box.

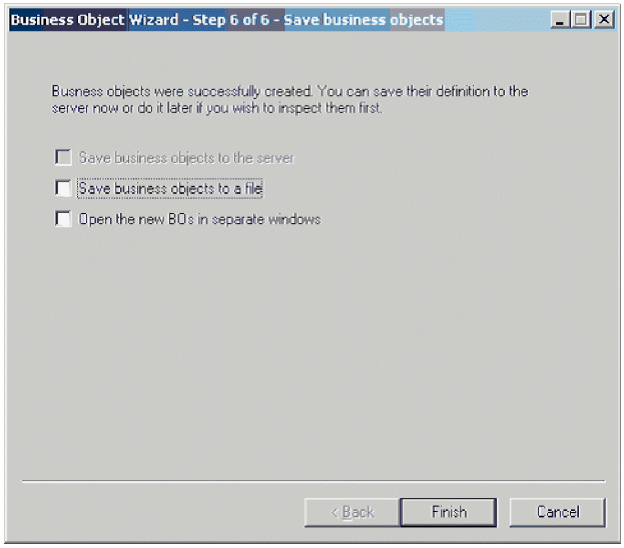


Figure 24. Saving the business object definition

Contents of the generated definition

The business object definition that PortalODA generates contains:

- An attribute for each column in the specified database tables and views
- The verbs specified in the BO Properties window (Figure 23 on page 58)
- Application-specific information:
  - At the business-object level
  - For each attribute
  - For each verb

When generating business objects by getting the information from Portal Infranet, the application-specific information generated is for simple attributes only. The exception to this rule is if the container attribute is a multi-value link. In all other cases, the user must enter the application-specific information as described in Chapter 3, "Understanding business objects," on page 25.

This section describes:

- "Business-object-level properties" on page 59
- "Attribute properties" on page 60
- "Verbs" on page 61

Business-object-level properties

PortalODA 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 contains the name of the corresponding Portal Infranet business component.

Attribute properties

This section describes the properties that PortalODA generates for each attribute.

Important: Any user edits described in the following sections refer to business object generation only, not to business object conversion.

Name property

PortalODA obtains the value of the attribute's name from the corresponding attribute in the Portal Infranet business component.

Data type property

When setting the type of an attribute, PortalODA converts the data type of the attribute in the Portal Infranet business component and converts it to the corresponding data type, as shown in Table 10. This is only in the case of business object generation, since business object conversion is for existing business objects.

Table 10. Correspondence of data types

Application	WebSphere business integration system	Length
PIN_FLDT_INT	Integer	
PIN_FLDT_ENUM	Integer	
PIN_FLDT_STR	String	Length of corresponding attribute in Portal Infranet
PIN_FLDT_BUF	String	Length of corresponding attribute in Portal Infranet
PIN_FLDT_POID	String	Length of corresponding attribute in Portal Infranet
PIN_FLDT_TSTAMP	Date	
PIN_FLDT_ARRAY	Object	
PIN_FLDT_SUBSTRUCT	Object	
PIN_FLDT_BINSTR	String	Length of corresponding attribute in Portal Infranet
PIN_FLDT_DECIMAL	Float	

Note: If an attribute's data type is not one of those shown in Table 10, PortalODA skips the column and displays a message stating that the column cannot be processed.

Cardinality property

PortalODA sets the cardinality of all simple attributes to 1 and the container attributes to `n`. The user should change the cardinality of the container attributes wherever it is needed.

MaxLength property

PortalODA obtains the length of the attribute from Portal Infranet.

IsKey property

PortalODA does not mark any attributes as key fields. You must manually mark the key fields after the business objects are generated.

IsRequired Property

If a field is designated not null in the table or view, PortalODA marks it as a required attribute. However, PortalODA does not mark the key field as required because the Portal Infranet application generates its own Id values while creating a record.

AppSpecificInfo Property

The user should edit this property if container attributes have not been generated and ensure the correctness if container attributes have been generated.

Verbs

PortalODA generates the verbs specified in the BO Properties window (as illustrated in Figure 23 on page 58). It creates an AppSpecificInfo property for each verb but does not populate it.

Adding information to the business object definition

Since Portal Infranet storable classes may not have all the information that a business objects requires, it may be necessary to add information to the business object definition that PortalODA creates, especially when generating new business objects.

To examine the business object definition or reload a revised definition into the repository, use Business Object Designer.

Note: Alternatively, if ICS is the integration broker, you can use the repos_copy command to load the definition into the repository; if WebSphere MQ Integrator Broker is the integration broker, you can use a system command to copy the file into the repository directory.

Appendix A. Standard connector properties

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

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

If your adapter supports DB2 Information Integrator, use the WMQI options and the DB2 II standard properties (see the Notes column in Table 11 on page 65.)

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

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

New properties

This standard property was added in this release:

- BOTrace

Standard connector properties overview

Connectors have two types of configuration properties:

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

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

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

Starting Connector Configurator

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

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

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

Configuration property values overview

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

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

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

A connector obtains its configuration values at startup. If you change the value of one or more connector properties during a run-time session, the property's update method determines how the change takes effect.

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

There are four update methods for standard connector properties:

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

To determine how a specific property is updated, refer to the **Update Method** column in the Connector Configurator window, or see the Update Method column in Table 11 on page 65.

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

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

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

Standard properties quick-reference

Table 11 provides a quick-reference to the standard connector configuration properties. Not all connectors require all of these properties, and property settings may differ from integration broker to integration broker.

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

Note: In the Notes column in Table 11, the phrase "RepositoryDirectory is set to <REMOTE>" indicates that the broker is InterChange Server. When the broker is WMQI or WAS, the repository directory is set to *<ProductDir>*\repository

Table 11. Summary of standard configuration properties

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

Table 11. Summary of standard configuration properties (continued)

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

Table 11. Summary of standard configuration properties (continued)

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

Table 11. Summary of standard configuration properties (continued)

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

Table 11. Summary of standard configuration properties (continued)

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

Table 11. Summary of standard configuration properties (continued)

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

Table 11. Summary of standard configuration properties (continued)

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

Standard properties

This section describes the standard connector configuration properties.

AdapterHelpName

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

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

AdminInQueue

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

The default value is <*CONNECTORNAME*>/ADMININQUEUE

AdminOutQueue

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

The default value is <*CONNECTORNAME*>/ADMINOUTQUEUE

AgentConnections

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

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

The default value of this property is 1.

AgentTraceLevel

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

The default value is 0.

ApplicationName

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

The default is the name of the connector.

BiDi.Application

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

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

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

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

BiDi.Broker

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

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

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

BiDi.Metadata

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

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

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

BiDi.Transformation

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

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

The default value is `false`.

BOTrace

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

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

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

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

The default value is `false`.

BrokerType

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

CharacterEncoding

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

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

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

CommonEventInfrastructure

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

The default value is `false`.

CommonEventInfrastructureContextURL

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

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

The default value is a blank field.

ConcurrentEventTriggeredFlows

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

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

To implement concurrent processing for an entire flow (from a source application to a destination application), the following properties must configured:

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

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

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

The default value is 1.

ContainerManagedEvents

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

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

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

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

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

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

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

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

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

There is no default value.

ControllerEventSequencing

The ControllerEventSequencing property enables event sequencing in the connector controller.

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

The default value is `true`.

ControllerStoreAndForwardMode

The ControllerStoreAndForwardMode property sets the behavior of the connector controller after it detects that the destination application-specific component is unavailable.

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

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

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

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

The default value is `true`.

ControllerTraceLevel

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

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

The default value is `0`.

DeliveryQueue

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

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

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

DeliveryTransport

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

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

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

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

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

WebSphere MQ and IDL

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

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

JMS

The JMS transport mechanism enables communication between the connector and client connector framework using Java Messaging Service (JMS).

If you select JMS as the delivery transport, additional JMS properties such as jms.MessageBrokerName, jms.FactoryClassName, jms.Password, and jms.UserName

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

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

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

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

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

  This script is located in the \bin directory below the product directory (<*ProductDir*>). Using a text editor, add the following line as the first line in the CWSharedEnv.sh script:

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

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

DuplicateEventElimination

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

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

The default value is `false`.

EnableOidForFlowMonitoring

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

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

The default value is `false`.

FaultQueue

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

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

jms.FactoryClassName

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

The default is CxCommon.Messaging.jms.IBMMQSeriesFactory.

jms.ListenerConcurrency

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

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

The default value is 1.

jms.MessageBrokerName

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

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

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

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

jms.NumConcurrentRequests

The jms.NumConcurrentRequests property specifies the maximum number of concurrent service call requests that can be sent to a connector at the same time. Once that maximum is reached, new service calls are blocked and must wait for another request to complete before proceeding.

The default value is 10.

jms.Password

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

There is no default value.

jms.TransportOptimized

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

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

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

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

The default value is `false`.

jms.UserName

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

There is no default value.

JvmMaxHeapSize

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

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

The default value is 128m.

JvmMaxNativeStackSize

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

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

The default value is 128k.

JvmMinHeapSize

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

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

The default value is 1m.

ListenerConcurrency

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

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

The default value is 1.

Locale

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

A locale name has the following format:

ll_TT.codeset

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

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

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

The default value is `en_US`.

LogAtInterchangeEnd

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

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

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

The default value is `false`.

MaxEventCapacity

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

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

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

The default value is 2147483647.

MessageFileName

The MessageFileName property specifies the name of the connector message file. The standard location for the message file is \connectors\messages in the product directory. Specify the message file name in an absolute path if the message file is not located in the standard location.

If a connector message file does not exist, the connector uses `InterchangeSystem.txt` as the message file. This file is located in the product directory.

Note: To determine whether a connector has its own message file, see the individual adapter user guide.

The default value is `InterchangeSystem.txt`.

MonitorQueue

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

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

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

OADAutoRestartAgent

the OADAutoRestartAgent property specifies whether the connector uses the automatic and remote restart feature. This feature uses the WebSphere MQ-triggered Object Activation Daemon (OAD) to restart the connector after an abnormal shutdown, or to start a remote connector from System Monitor.

This property must be set to `true` to enable the automatic and remote restart feature. For information on how to configure the WebSphere MQ-triggered OAD feature. see the *Installation Guide for Windows* or *for UNIX*.

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

The default value is `false`.

OADMaxNumRetry

The OADMaxNumRetry property specifies the maximum number of times that the WebSphere MQ-triggered Object Activation Daemon (OAD) automatically attempts to restart the connector after an abnormal shutdown. The OADAutoRestartAgent property must be set to `true` for this property to take effect.

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

The default value is 1000.

OADRetryTimeInterval

The OADRetryTimeInterval property specifies the number of minutes in the retry-time interval for the WebSphere MQ-triggered Object Activation Daemon (OAD). If the connector agent does not restart within this retry-time interval, the connector controller asks the OAD to restart the connector agent again. The OAD repeats this retry process as many times as specified by the OADMaxNumRetry property. The OADAutoRestartAgent property must be set to true for this property to take effect.

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

The default value is 10.

PollEndTime

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

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

If the adapter runtime detects:

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

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

PollFrequency

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

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

The following values are valid for this property:

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

The default is 10000.

Important: Some connectors have restrictions on the use of this property. Where they exist, these restrictions are documented in the chapter on installing and configuring the adapter.

PollQuantity

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

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

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

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

PollStartTime

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

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

If the adapter runtime detects:

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

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

RepositoryDirectory

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

If the integration broker is ICS, this value must be set to set to <REMOTE> because the connector obtains this information from the InterChange Server repository.

When the integration broker is a WebSphere message broker or WAS, this value is set to <*ProductDir*>\repository by default. However, it may be set to any valid directory name.

RequestQueue

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

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

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

ResponseQueue

The ResponseQueue property specifies the JMS response queue, which delivers a response message from the connector framework to the integration broker. When the integration broker is ICS, the server sends the request and waits for a response message in the JMS response queue.

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

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

RestartRetryCount

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

The default value is 7.

RestartRetryInterval

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

Possible values for the property range from 1 through 2147483647.

The default value is 1.

ResultsSetEnabled

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

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

The default value is false.

ResultsSetSize

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

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

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

RHF2MessageDomain

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

This is an example header:

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

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

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

SourceQueue

The SourceQueue property designates the JMS source queue for the connector framework in support of guaranteed event delivery for JMS-enabled connectors that use a JMS event store. For further information, see "ContainerManagedEvents" on page 74.

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

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

SynchronousRequestQueue

The SynchronousRequestQueue property delivers request messages that require a synchronous response from the connector framework to the broker. This queue is necessary only if the connector uses synchronous execution. With synchronous execution, the connector framework sends a message to the synchronous request queue and waits for a response from the broker on the synchronous response queue. The response message sent to the connector has a correlation ID that matches the ID of the original message.

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

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

SynchronousRequestTimeout

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

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

The default value is `0`.

SynchronousResponseQueue

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

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

The default is `<CONNECTORNAME>/SYNCHRONOUSRESPONSEQUEUE`

TivoliMonitorTransactionPerformance

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

The default value is `false`.

WireFormat

The WireFormat property specifies the message format on the transport:

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

WsifSynchronousRequestTimeout

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

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

The default value is 0.

XMLNameSpaceFormat

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

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

The default value is `short`.

Appendix B. Using Connector Configurator

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

You use Connector Configurator to:

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

The topics covered in this appendix are:

Overview of Connector Configurator

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

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

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

You use Connector Configurator to:

- Create a **connector-specific property template** for configuring your connector.
- Create a **connector configuration file**; you must create one configuration file for each connector you install.
- Set properties in a configuration file.
  You may need to modify the default values that are set for properties in the connector templates. You must also designate supported business object definitions and, with ICS, maps for use with collaborations as well as specify messaging, logging and tracing, and data handler parameters, as required.

The mode in which you run Connector Configurator, and the configuration file type you use, may differ according to which integration broker you are running. For example, if WMQI is your broker, you run Connector Configurator directly, and not from within System Manager (see "Running Configurator in stand-alone mode" on page 88).

Connector configuration properties include both standard configuration properties (the properties that all connectors have) and connector-specific properties (properties that are needed by the connector for a specific application or technology).

Because **standard properties** are used by all connectors, you do not need to define those properties from scratch; Connector Configurator incorporates them into your configuration file as soon as you create the file. However, you do need to set the value of each standard property in Connector Configurator.

The range of standard properties may not be the same for all brokers and all configurations. Some properties are available only if other properties are given a specific value. The Standard Properties window in Connector Configurator will show the properties available for your particular configuration.

For **connector-specific properties**, however, you need first to define the properties and then set their values. You do this by creating a connector-specific property template for your particular adapter. There may already be a template set up in your system, in which case, you simply use that. If not, follow the steps in "Creating a new template" on page 89 to set up a new one.

Running connectors on UNIX

Connector Configurator runs only in a Windows environment. If you are running the connector in a UNIX environment, use Connector Configurator in Windows to modify the configuration file and then copy the file to your UNIX environment.

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

Starting Connector Configurator

You can start and run Connector Configurator in either of two modes:

- Independently, in stand-alone mode
- From System Manager

Running Configurator in stand-alone mode

You can run Connector Configurator without running System Manager and work with connector configuration files, irrespective of your broker.

To do so:

- From **Start>Programs**, click **IBM WebSphere Business Integration Adapters>IBM WebSphere Business Integration Toolset>Connector Configurator**.
- Select **File>New>Connector Configuration**.
- When you click the pull-down menu next to **System Connectivity Integration Broker**, you can select ICS, WebSphere Message Brokers or WAS, depending on your broker.

You may choose to run Connector Configurator independently to generate the file, and then connect to System Manager to save it in a System Manager project (see "Completing a configuration file" on page 94.)

Running Configurator from System Manager

You can run Connector Configurator from System Manager.

To run Connector Configurator:

1. Open the System Manager.
2. In the System Manager window, expand the **Integration Component Libraries** icon and highlight **Connectors**.
3. From the System Manager menu bar, click **Tools>Connector Configurator**. The Connector Configurator window opens and displays a **New Connector** dialog box.
4. When you click the pull-down menu next to **System Connectivity Integration Broker**, you can select ICS, WebSphere Message Brokers or WAS, depending on your broker.

To edit an existing configuration file:

- In the System Manager window, select any of the configuration files listed in the Connector folder and right-click on it. Connector Configurator opens and displays the configuration file with the integration broker type and file name at the top.
- From Connector Configurator, select **File>Open**. Select the name of the connector configuration file from a project or from the directory in which it is stored.
- Click the Standard Properties tab to see which properties are included in this configuration file.

Creating a connector-specific property template

To create a configuration file for your connector, you need a connector-specific property template as well as the system-supplied standard properties.

You can create a brand-new template for the connector-specific properties of your connector, or you can use an existing connector definition as the template.

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

Creating a new template

This section describes how you create properties in the template, define general characteristics and values for those properties, and specify any dependencies between the properties. Then you save the template and use it as the base for creating a new connector configuration file.

To create a template in Connector Configurator:

1. Click **File>New>Connector-Specific Property Template**.
2. The **Connector-Specific Property Template** dialog box appears.
   - Enter a name for the new template in the **Name** field below **Input a New Template Name.** You will see this name again when you open the dialog box for creating a new configuration file from a template.

- To see the connector-specific property definitions in any template, select that template's name in the **Template Name** display. A list of the property definitions contained in that template appears in the **Template Preview** display.

3. You can use an existing template whose property definitions are similar to those required by your connector as a starting point for your template. If you do not see any template that displays the connector-specific properties used by your connector, you will need to create one.
   - If you are planning to modify an existing template, select the name of the template from the list in the **Template Name** table below **Select the Existing Template to Modify: Find Template.**
   - This table displays the names of all currently available templates. You can also search for a template.

Specifying general characteristics

When you click **Next** to select a template, the **Properties - Connector-Specific Property Template** dialog box appears. The dialog box has tabs for General characteristics of the defined properties and for Value restrictions. The General display has the following fields:

- **General:**
  Property Type
  Property Subtype
  Updated Method
  Description
- **Flags**
  Standard flags
- **Custom Flag**
  Flag

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

After you have made selections for the general characteristics of the property, click the **Value** tab.

Specifying values

The **Value** tab enables you to set the maximum length, the maximum multiple values, a default value, or a value range for the property. It also allows editable values. To do so:

1. Click the **Value** tab. The display panel for Value replaces the display panel for General.
2. Select the name of the property in the **Edit properties** display.
3. In the fields for **Max Length** and **Max Multiple Values**, enter your values.

To create a new property value:

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

The **Value** panel displays a table with three columns:

The **Value** column shows the value that you entered in the **Property Value** dialog box, and any previous values that you created.

The **Default Value** column allows you to designate any of the values as the default.

The **Value Range** shows the range that you entered in the **Property Value** dialog box.

After a value has been created and appears in the grid, it can be edited from within the table display.

To make a change in an existing value in the table, select an entire row by clicking on the row number. Then right-click in the **Value** field and click **Edit Value**.

Setting dependencies

When you have made your changes to the **General** and **Value** tabs, click **Next**. The **Dependencies - Connector-Specific Property Template** dialog box appears.

A dependent property is a property that is included in the template and used in the configuration file *only if* the value of another property meets a specific condition. For example, `PollQuantity` appears in the template only if JMS is the transport mechanism and `DuplicateEventElimination` is set to `True`.
To designate a property as dependent and to set the condition upon which it depends, do this:

1. In the **Available Properties** display, select the property that will be made dependent.
2. In the **Select Property** field, use the drop-down menu to select the property that will hold the conditional value.
3. In the **Condition Operator** field, select one of the following:

   == (equal to)

   != (not equal to)

   > (greater than)

   < (less than)

   >= (greater than or equal to)

   <=(less than or equal to)
4. In the **Conditional Value** field, enter the value that is required in order for the dependent property to be included in the template.
5. With the dependent property highlighted in the **Available Properties** display, click an arrow to move it to the **Dependent Property** display.
6. Click **Finish**. Connector Configurator stores the information you have entered as an XML document, under `\data\app` in the `\bin` directory where you have installed Connector Configurator.

Setting pathnames

Some general rules for setting pathnames are:

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

- Queue names may not have leading or embedded spaces.

Creating a new configuration file

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

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

To start Connector Configurator:

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

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

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

Then select **File>New>Connector Configuration**. In the New Connector window, enter the name of the new connector.

You also need to select an integration broker. The broker you select determines the properties that will appear in the configuration file. To select a broker:

- In the **Integration Broker** field, select ICS, WebSphere Message Brokers or WAS connectivity.
- Complete the remaining fields in the **New Connector** window, as described later in this chapter.

Creating a configuration file from a connector-specific template

Once a connector-specific template has been created, you can use it to create a configuration file:

1. Set the operating system for extended validation of the configuration file using the **Target System:** droplist on the menu bar (see "Creating a new configuration file" above).
2. Click **File>New>Connector Configuration**.
3. The **New Connector** dialog box appears, with the following fields:
   - **Name**

     Enter the name of the connector. Names are case-sensitive. The name you enter must be unique, and must be consistent with the file name for a connector that is installed on the system.

     Important: Connector Configurator does not check the spelling of the name that you enter. You must ensure that the name is correct.
   - **System Connectivity**

     Click ICS or WebSphere Message Brokers or WAS.
   - **Select Connector-Specific Property Template**

Type the name of the template that has been designed for your connector. The available templates are shown in the **Template Name** display. When you select a name in the Template Name display, the **Property Template Preview** display shows the connector-specific properties that have been defined in that template.

Select the template you want to use and click **OK**.

4. A configuration screen appears for the connector that you are configuring. The title bar shows the integration broker and connector name. You can fill in all the field values to complete the definition now, or you can save the file and complete the fields later.
5. To save the file, click **File>Save>To File** or **File>Save>To Project**. To save to a project, System Manager must be running.
   If you save as a file, the **Save File Connector** dialog box appears. Choose `*.cfg` as the file type, verify in the File Name field that the name is spelled correctly and has the correct case, navigate to the directory where you want to locate the file, and click **Save**. The status display in the message panel of Connector Configurator indicates that the configuration file was successfully created.

   Important: The directory path and name that you establish here must match the connector configuration file path and name that you supply in the startup file for the connector.
6. To complete the connector definition, enter values in the fields for each of the tabs of the Connector Configurator window, as described later in this chapter.

Using an existing file

You may have an existing file available in one or more of the following formats:

- A connector definition file.
  This is a text file that lists properties and applicable default values for a specific connector. Some connectors include such a file in a `\repository` directory in their delivery package (the file typically has the extension `.txt`; for example, CN_XML.txt for the XML connector).
- An ICS repository file.
  Definitions used in a previous ICS implementation of the connector may be available to you in a repository file that was used in the configuration of that connector. Such a file typically has the extension `.in` or `.out`.
- A previous configuration file for the connector.
  Such a file typically has the extension *.cfg.

Although any of these file sources may contain most or all of the connector-specific properties for your connector, the connector configuration file will not be complete until you have opened the file and set properties, as described later in this chapter.

To use an existing file to configure a connector, you must open the file in Connector Configurator, revise the configuration, and then resave the file.

Follow these steps to open a *.txt, *.cfg, or *.in file from a directory:

1. In Connector Configurator, click **File>Open>From File**.
2. In the **Open File Connector** dialog box, select one of the following file types to see the available files:
   - Configuration (`*.cfg`)
   - ICS Repository (`*.in`, `*.out`)

Choose this option if a repository file was used to configure the connector in an ICS environment. A repository file may include multiple connector definitions, all of which will appear when you open the file.

- All files (*.*)

  Choose this option if a `*.txt` file was delivered in the adapter package for the connector, or if a definition file is available under another extension.

3. In the directory display, navigate to the appropriate connector definition file, select it, and click **Open**.

Follow these steps to open a connector configuration from a System Manager project:

1. Start System Manager. A configuration can be opened from or saved to System Manager only if System Manager has been started.
2. Start Connector Configurator.
3. Click **File>Open>From Project**.

Completing a configuration file

When you open a configuration file or a connector from a project, the Connector Configurator window displays the configuration screen, with the current attributes and values.

The title of the configuration screen displays the integration broker and connector name as specified in the file. Make sure you have the correct broker. If not, change the broker value before you configure the connector. To do so:

1. Under the **Standard Properties** tab, select the value field for the `BrokerType` property. In the drop-down menu, select the value `ICS`, `WMQI`, or `WAS`.
2. The Standard Properties tab will display the connector properties associated with the selected broker. The table shows **Property name**, **Value**, **Type**, **Subtype** (if the Type is a string), **Description**, and **Update Method**.
3. You can save the file now or complete the remaining configuration fields, as described in "Specifying supported business object definitions" on page 97..
4. When you have finished your configuration, click **File>Save>To Project** or **File>Save>To File**.

   If you are saving to file, select `*.cfg` as the extension, select the correct location for the file and click **Save**.

   If multiple connector configurations are open, click **Save All to File** to save all of the configurations to file, or click **Save All to Project** to save all connector configurations to a System Manager project.

   Before you created the configuration file, you used the **Target System** droplist that allows you to select the target operating system for extended validation of the properties.

   Before it saves the file, Connector Configurator checks that values have been set for all required standard properties. If a required standard property is missing a value, Connector Configurator displays a message that the validation failed. You must supply a value for the property in order to save the configuration file.

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

Setting the configuration file properties

When you create and name a new connector configuration file, or when you open an existing connector configuration file, Connector Configurator displays a configuration screen with tabs for the categories of required configuration values.

Connector Configurator requires values for properties in these categories for connectors running on all brokers:

- Standard Properties
- Connector-specific Properties
- Supported Business Objects
- Trace/Log File values
- Data Handler (applicable for connectors that use JMS messaging with guaranteed event delivery)

Note: For connectors that use JMS messaging, an additional category may display, for configuration of data handlers that convert the data to business objects.

For connectors running on **ICS**, values for these properties are also required:

- Associated Maps
- Resources
- Messaging (where applicable)
- Security

Important: Connector Configurator accepts property values in either English or non-English character sets. However, the names of both standard and connector-specific properties, and the names of supported business objects, must use the English character set only.

Standard properties differ from connector-specific properties as follows:

- Standard properties of a connector are shared by both the application-specific component of a connector and its broker component. All connectors have the same set of standard properties. These properties are described in Appendix A of each adapter guide. You can change some but not all of these values.
- Application-specific properties apply only to the application-specific component of a connector, that is, the component that interacts directly with the application. Each connector has application-specific properties that are unique to its application. Some of these properties provide default values and some do not; you can modify some of the default values. The installation and configuration chapters of each adapter guide describe the application-specific properties and the recommended values.

The fields for **Standard Properties** and **Connector-Specific Properties** are color-coded to show which are configurable:

- A field with a grey background indicates a standard property. You can change the value but cannot change the name or remove the property.
- A field with a white background indicates an application-specific property. These properties vary according to the specific needs of the application or connector. You can change the value and delete these properties.
- Value fields are configurable.

- The **Update Method** field is displayed for each property. It indicates whether a component or agent restart is necessary to activate changed values. You cannot configure this setting.

Setting standard connector properties

To change the value of a standard property:

1. Click in the field whose value you want to set.
2. Either enter a value, or select one from the drop-down menu if it appears.

   Note: If the property has a Type of String, it may have a subtype value in the Subtype column. This subtype is used for extended validation of the property.
3. After entering all the values for the standard properties, you can do one of the following:
   - To discard the changes, preserve the original values, and exit Connector Configurator, click **File>Exit** (or close the window), and click **No** when prompted to save changes.
   - To enter values for other categories in Connector Configurator, select the tab for the category. The values you enter for **Standard Properties** (or any other category) are retained when you move to the next category. When you close the window, you are prompted to either save or discard the values that you entered in all the categories as a whole.
   - To save the revised values, click **File>Exit** (or close the window) and click **Yes** when prompted to save changes. Alternatively, click **Save>To File** from either the File menu or the toolbar.

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

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

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

Setting connector-specific configuration properties

For connector-specific configuration properties, you can add or change property names, configure values, delete a property, and encrypt a property. The default property length is 255 characters.

1. Right-click in the top left portion of the grid. A pop-up menu bar will appear. Click **Add** to add a property. To add a child property, right-click on the parent row number and click **Add child**.
2. Enter a value for the property or child property.

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

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

   Note: If the hot button does not appear, no Extended Help was found for that property.
5. Choose to save or discard changes, as described for "Setting standard connector properties" on page 96.

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

The Update Method displayed for each property indicates whether a component or agent restart is necessary to activate changed values.

Important: Changing a preset application-specific connector property name may cause a connector to fail. Certain property names may be needed by the connector to connect to an application or to run properly.

Encryption for connector properties

Application-specific properties can be encrypted by selecting the **Encrypt** check box in the Connector-specific Properties window. To decrypt a value, click to clear the **Encrypt** check box, enter the correct value in the **Verification** dialog box, and click **OK**. If the entered value is correct, the value is decrypted and displays.

The adapter user guide for each connector contains a list and description of each property and its default value.

If a property has multiple values, the **Encrypt** check box will appear for the first value of the property. When you select **Encrypt**, all values of the property will be encrypted. To decrypt multiple values of a property, click to clear the **Encrypt** check box for the first value of the property, and then enter the new value in the **Verification** dialog box. If the input value is a match, all multiple values will decrypt.

Update method

Refer to the descriptions of update methods found in the Standard Properties appendix, under "Configuration property values overview" on page 64.

Specifying supported business object definitions

Use the **Supported Business Objects** tab in Connector Configurator to specify the business objects that the connector will use. You must specify both generic business objects and application-specific business objects, and you must specify associations for the maps between the business objects.

Note: Some connectors require that certain business objects be specified as supported in order to perform event notification or additional configuration

(using meta-objects) with their applications. For more information, see the *Connector Development Guide for C++* or the *Connector Development Guide for Java*.

If ICS is your broker

To specify that a business object definition is supported by the connector, or to change the support settings for an existing business object definition, click the **Supported Business Objects** tab and use the following fields.

Business object name: To designate that a business object definition is supported by the connector, with System Manager running:

1. Click an empty field in the **Business Object Name** list. A drop list displays, showing all the business object definitions that exist in the System Manager project.
2. Click on a business object to add it.
3. Set the **Agent Support** (described below) for the business object.
4. In the File menu of the Connector Configurator window, click **Save to Project**. The revised connector definition, including designated support for the added business object definition, is saved to an ICL (Integration Component Library) project in System Manager.

To delete a business object from the supported list:

1. To select a business object field, click the number to the left of the business object.
2. From the **Edit** menu of the Connector Configurator window, click **Delete Row**. The business object is removed from the list display.
3. From the **File** menu, click **Save to Project**.

Deleting a business object from the supported list changes the connector definition and makes the deleted business object unavailable for use in this implementation of this connector. It does not affect the connector code, nor does it remove the business object definition itself from System Manager.

Agent support: If a business object has Agent Support, the system will attempt to use that business object for delivering data to an application via the connector agent.

Typically, application-specific business objects for a connector are supported by that connector's agent, but generic business objects are not.

To indicate that the business object is supported by the connector agent, check the **Agent Support** box. The Connector Configurator window does not validate your Agent Support selections.

Maximum transaction level: The maximum transaction level for a connector is the highest transaction level that the connector supports.

For most connectors, `Best Effort` is the only possible choice.

You must restart the server for changes in transaction level to take effect.

If a WebSphere Message Broker is your broker

If you are working in stand-alone mode (not connected to System Manager), you must enter the business object name manually.

If you have System Manager running, you can select the empty box under the **Business Object Name** column in the **Supported Business Objects** tab. A combo box appears with a list of the business object available from the Integration Component Library project to which the connector belongs. Select the business object you want from the list.

The **Message Set ID** is an optional field for WebSphere Business Integration Message Broker 5.0, and need not be unique if supplied. However, for WebSphere MQ Integrator and Integrator Broker 2.1, you must supply a unique **ID**.

If WAS is your broker

When WebSphere Application Server is selected as your broker type, Connector Configurator does not require message set IDs. The **Supported Business Objects** tab shows a **Business Object Name** column only for supported business objects.

If you are working in stand-alone mode (not connected to System Manager), you must enter the business object name manually.

If you have System Manager running, you can select the empty box under the Business Object Name column in the Supported Business Objects tab. A combo box appears with a list of the business objects available from the Integration Component Library project to which the connector belongs. Select the business object you want from this list.

Associated maps (ICS)

Each connector supports a list of business object definitions and their associated maps that are currently active in WebSphere InterChange Server. This list appears when you select the **Associated Maps** tab.

The list of business objects contains the application-specific business object which the agent supports and the corresponding generic object that the controller sends to the subscribing collaboration. The association of a map determines which map will be used to transform the application-specific business object to the generic business object or the generic business object to the application-specific business object.

If you are using maps that are uniquely defined for specific source and destination business objects, the maps will already be associated with their appropriate business objects when you open the display, and you will not need (or be able) to change them.

If more than one map is available for use by a supported business object, you will need to explicitly bind the business object with the map that it should use.

The **Associated Maps** tab displays the following fields:

- **Business Object Name**

  These are the business objects supported by this connector, as designated in the **Supported Business Objects** tab. If you designate additional business objects under the Supported Business Objects tab, they will be reflected in this list after you save the changes by choosing **Save to Project** from the **File** menu of the Connector Configurator window.
- **Associated Maps**

The display shows all the maps that have been installed to the system for use with the supported business objects of the connector. The source business object for each map is shown to the left of the map name, in the **Business Object Name** display.

- **Explicit Binding**

  In some cases, you may need to explicitly bind an associated map.

  Explicit binding is required only when more than one map exists for a particular supported business object. When ICS boots, it tries to automatically bind a map to each supported business object for each connector. If more than one map takes as its input the same business object, the server attempts to locate and bind one map that is the superset of the others.

  If there is no map that is the superset of the others, the server will not be able to bind the business object to a single map, and you will need to set the binding explicitly.

  To explicitly bind a map:

  1. In the **Explicit** column, place a check in the check box for the map you want to bind.
  2. Select the map that you intend to associate with the business object.
  3. In the **File** menu of the Connector Configurator window, click **Save to Project**.
  4. Deploy the project to ICS.
  5. Reboot the server for the changes to take effect.

Resources (ICS)

The **Resource** tab allows you to set a value that determines whether and to what extent the connector agent will handle multiple processes concurrently, using connector agent parallelism.

Not all connectors support this feature. If you are running a connector agent that was designed in Java to be multi-threaded, you are advised not to use this feature, since it is usually more efficient to use multiple threads than multiple processes.

Messaging (ICS)

The **Messaging** tab enables you to configure messaging properties. The messaging properties are available only if you have set MQ as the value of the `DeliveryTransport` standard property and ICS as the broker type. These properties affect how your connector will use queues.

Validating messaging queues

Before you can validate a messaging queue, you must:

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

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

Security (ICS)

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

Setting trace/log file values

When you open a connector configuration file or a connector definition file, Connector Configurator uses the logging and tracing values of that file as default values. You can change those values in Connector Configurator.

To change the logging and tracing values:

1. Click the **Trace/Log Files** tab.
2. For either logging or tracing, you can choose to write messages to one or both of the following:
   - To console (STDOUT):
     Writes logging or tracing messages to the STDOUT display.

     Note: You can only use the STDOUT option from the **Trace/Log Files** tab for connectors running on the Windows platform.
   - To File:
     Writes logging or tracing messages to a file that you specify. To specify the file, click the directory button (ellipsis), navigate to the preferred location, provide a file name, and click **Save**. Logging or tracing message are written to the file and location that you specify.

     Note: Both logging and tracing files are simple text files. You can use the file extension that you prefer when you set their file names. For tracing files, however, it is advisable to use the extension `.trace` rather than `.trc`, to avoid confusion with other files that might reside on the system. For logging files, `.log` and `.txt` are typical file extensions.

Data handlers

The data handlers section is available for configuration only if you have designated a value of JMS for DeliveryTransport and a value of JMS for ContainerManagedEvents. Not all adapters make use of data handlers.

See the descriptions under `ContainerManagedEvents` in Appendix A, Standard Properties, for values to use for these properties. For additional details, see the *Connector Development Guide for C++* or the *Connector Development Guide for Java*.

Saving your configuration file

When you have finished configuring your connector, save the connector configuration file. Connector Configurator saves the file in the broker mode that you selected during configuration. The title bar of Connector Configurator always displays the broker mode (ICS, WMQI or WAS) that it is currently using.

The file is saved as an XML document. You can save the XML document in three ways:

- From System Manager, as a file with a `*.con` extension in an Integration Component Library, or
- In a directory that you specify.
- In stand-alone mode, as a file with a `*.cfg` extension in a directory folder. By default, the file is saved to `\WebSphereAdapters\bin\Data\App`.
- You can also save it to a WebSphere Application Server project if you have set one up.

For details about using projects in System Manager, and for further information about deployment, see the following implementation guides:

- For ICS: *Implementation Guide for WebSphere InterChange Server*
- For WebSphere Message Brokers: *Implementing Adapters with WebSphere Message Brokers*
- For WAS: *Implementing Adapters with WebSphere Application Server*

Changing a configuration file

You can change the integration broker setting for an existing configuration file. This enables you to use the file as a template for creating a new configuration file, which can be used with a different broker.

Note: You will need to change other configuration properties as well as the broker mode property if you switch integration brokers.

To change your broker selection within an existing configuration file (optional):

- Open the existing configuration file in Connector Configurator.
- Select the **Standard Properties** tab.
- In the **BrokerType** field of the Standard Properties tab, select the value that is appropriate for your broker.
  When you change the current value, the available tabs and field selections in the properties window will immediately change, to show only those tabs and fields that pertain to the new broker you have selected.

Completing the configuration

After you have created a configuration file for a connector and modified it, make sure that the connector can locate the configuration file when the connector starts up.

To do so, open the startup file used for the connector, and verify that the location and file name used for the connector configuration file match exactly the name you have given the file and the directory or path where you have placed it.

Using Connector Configurator in a globalized environment

Connector Configurator is globalized and can handle character conversion between the configuration file and the integration broker. Connector Configurator uses native encoding. When it writes to the configuration file, it uses UTF-8 encoding.

Connector Configurator supports non-English characters in:

- All value fields
- Log file and trace file path (specified in the **Trace/Log files** tab)

The drop list for the `CharacterEncoding` and `Locale` standard configuration properties displays only a subset of supported values. To add other values to the drop list, you must manually modify the `\Data\Std\stdConnProps.xml` file in the product directory.

For example, to add the locale `en_GB` to the list of values for the `Locale` property, open the `stdConnProps.xml` file and add the line in boldface type below:

```
<Property name="Locale"
isRequired="true"
updateMethod="component restart">
                <ValidType>String</ValidType>
            <ValidValues>
                                <Value>ja_JP</Value>
                                <Value>ko_KR</Value>
                                <Value>zh_CN</Value>
                                <Value>zh_TW</Value>
                                <Value>fr_FR</Value>
                                <Value>de_DE</Value>
                                <Value>it_IT</Value>
                                <Value>es_ES</Value>
                                <Value>pt_BR</Value>
                                <Value>en_US</Value>
                                <Value>en_GB</Value>
                    <DefaultValue>en_US</DefaultValue>
            </ValidValues>
    </Property>
```

Appendix C. Application Response Measurement

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

Application Response Measurement instrumentation support

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

Required software

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

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

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

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

Enabling Application Response Measurement

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

Transaction monitoring

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

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

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

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

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

For more information

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

Appendix D. Common Event Infrastructure

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

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

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

Required software

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

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

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

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

Enabling Common Event Infrastructure

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

Obtaining Common Event Infrastructure adapter events

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

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

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

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

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

For more information

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

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

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

Common Event Infrastructure event catalog definitions

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

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

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

XML format for "start adapter" metadata

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

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

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

XML format for ″stop adapter″ metadata

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

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

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

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

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

XML format for “timeout adapter” metadata

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

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

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

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

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

XML format for ″request″ or ″delivery″ metadata

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

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

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

Index

Notices

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

IBM may not offer the products, services, or features discussed in this document in other countries. Consult your local IBM representative for information on the products and services currently available in your area. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any non-IBM product, program, or service.

IBM may have patents or pending patent applications covering subject matter described in this document. The furnishing of this document does not grant you any license to these patents. You can send license inquiries, in writing, to:

IBM Director of Licensing
IBM Corporation
North Castle Drive
Armonk, NY 10504-1785
U.S.A.

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

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

The following paragraph does not apply to the United Kingdom or any other country where such provisions are inconsistent with local law:

INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Some states do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement may not apply to you.

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time without notice.

Any references in this information to non-IBM Web sites are provided for convenience only and do not in any manner serve as an endorsement of those Web sites. The materials at those Web sites are not part of the materials for this IBM product and use of those Web sites is at your own risk.

IBM may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation to you.

Licensees of this program who wish to have information about it for the purpose of enabling: (i) the exchange of information between independently created programs and other programs (including this one) and (ii) the mutual use of the information which has been exchanged, should contact:

IBM Corporation
577 Airport Blvd., Suite 800
Burlingame, CA 94010
U.S.A.

Such information may be available, subject to appropriate terms and conditions, including in some cases, payment of a fee.

The licensed program described in this document and all licensed material available for it are provided by IBM under terms of the IBM Customer Agreement, IBM International Program License Agreement or any equivalent agreement between us.

Any performance data contained herein was determined in a controlled environment. Therefore, the results obtained in other operating environments may vary significantly. Some measurements may have been made on development-level systems and there is no guarantee that these measurements will be the same on generally available systems. Furthermore, some measurements may have been estimated through extrapolation. Actual results may vary. Users of this document should verify the applicable data for their specific environment.

Information concerning non-IBM products was obtained from the suppliers of those products, their published announcements or other publicly available sources. IBM has not tested those products and cannot confirm the accuracy of performance, compatibility or any other claims related to non-IBM products. Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products.

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

This information contains examples of data and reports used in daily business operations. To illustrate them as completely as possible, the examples include the names of individuals, companies, brands, and products. All of these names are fictitious and any similarity to the names and addresses used by an actual business enterprise is entirely coincidental.

COPYRIGHT LICENSE:

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

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

Programming interface information

Programming interface information, if provided, is intended to help you create application software using this program.

General-use programming interfaces allow you to write application software that obtain the services of this program's tools.

However, this information may also contain diagnosis, modification, and tuning information. Diagnosis, modification and tuning information is provided to help you debug your application software.

Warning: Do not use this diagnosis, modification, and tuning information as a programming interface because it is subject to change.

Trademarks and service marks

The following terms are trademarks or registered trademarks of International Business Machines Corporation in the United States or other countries, or both:

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

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

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

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

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

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

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

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

WebSphere Business Integration Adapter Framework, version 2.6.0.3

Printed in USA