IBM WebSphere Business Integration Server
Express and Express Plus

Adapter for Portal Infranet User Guide

Version 4.3

IBM WebSphere Business Integration Server
Express and Express Plus

Adapter for Portal Infranet User Guide

Version 4.3

Note!
Before using this information and the product it supports, read the information in “Notices” on page 93.“Notices.”

14 May 2004

This edition of this document applies to IBM WebSphere Business Integration Server Express, version 4.3, IBM Websphere Business Integration Server Express Plus, version 4.3, and to all subsequent releases and modifications until otherwise indicated in new editions.

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

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

Contents

About this document

The products IBM® WebSphere® Business Server Express and IBM® WebSphere® Business Server Express Plus are made up of the following components: InterChange Server Express, the associated Toolset Express, CollaborationFoundation, and a set of software integration adapters. The tools in the Toolset help you to create, modify, and manage business processes. You can choose from among the prepackaged adapters for your business processes that span applications. The standard processes template - CollaborationFoundation - allows you to quickly create customized processes.

Except where noted, all the information in this guide applies to both IBM WebSphere Business Integration Server Express and IBM WebSphere Business Integration Server Express Plus. The term WebSphere Business Integration Server Express and its variants refer to both products.

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 Server Express installations, and includes reference material on specific components.

You can download, install, and view the documentation at the following site: http://www.ibm.com/websphere/wbiserverexpress/infocenter

Note: Important information about this product may be available in Technical Support Technotes and Flashes issued after this document was published. These can be found on the WebSphere Business Integration Support Web site, http://www.ibm.com/software/integration/websphere/support/. Select the component area of interest and browse the Technotes and Flashes sections.

Typographic conventions

This document uses the following conventions:

`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. All WebSphere Business Integration Server Express 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.
ProductDir	Represents the directory where the product is installed. The directory for Business Integration Server Express is *\WebSphere Serve*r

New in this release

Release 4.3

First Paragraph

This is the first release of this guide.

Chapter 1. Overview of the adapter

This chapter describes the Adapter for the Portal Infranet component of WebSphere Business Integration Server Express and Server Express Plus, 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 InterChange Server Express and the application-specific component. The application-specific component contains code tailored to a particular application. The connector framework provides the following services between InterChange Server Express, 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 InterChange Server InterChange Server Express 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 InterChange Server Express 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 Server Express Adapter event facilities module: a C++ DLL on Windows that implements the event notification mechanism in the Infranet application. This module selects the Infranet events relevant to InterChange Server Express 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 InterChange Server Express. The connector also responds to business object requests from InterChange

Server Express. It generates logging and tracing messages that it writes to a file or the connector console, or sends to InterChange Server Express.

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

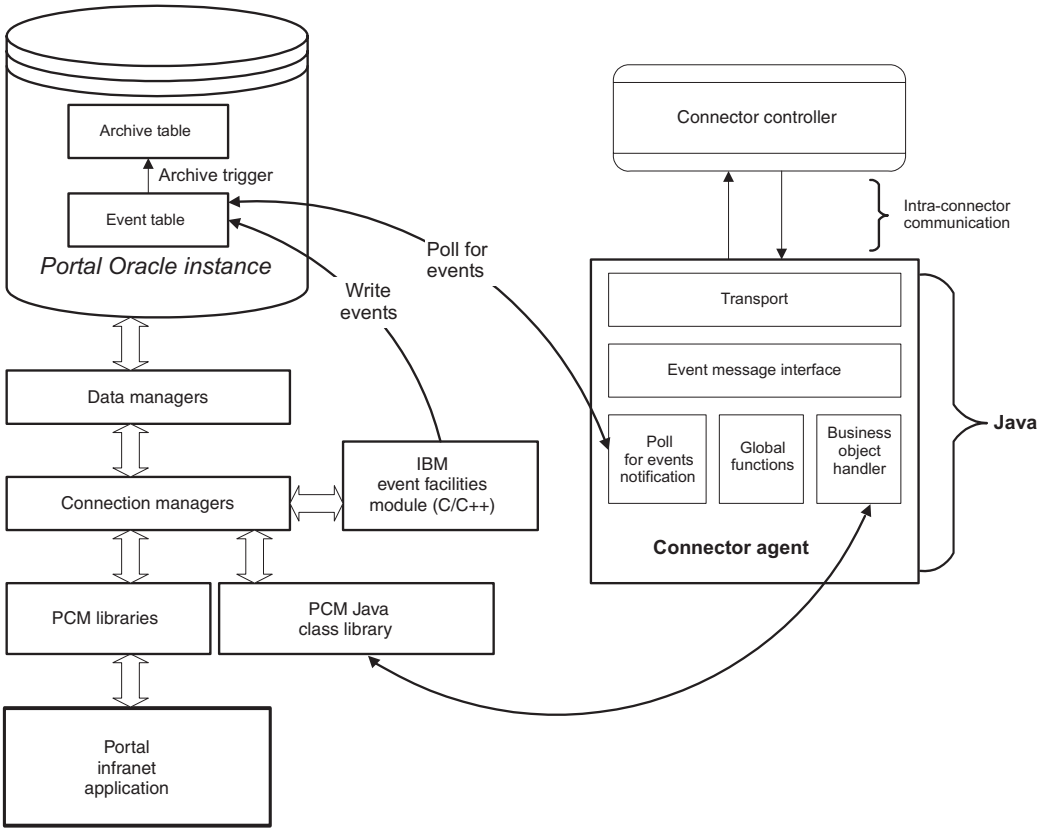


Figure 1. Connector architecture

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 machine for Windows. This API is used by the business object handler in the connector to exchange information between the adapter, InterChange Server Express, and Portal. The WebSphere Business Integration Server Express 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 27.

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 InterChange Server Express.

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.

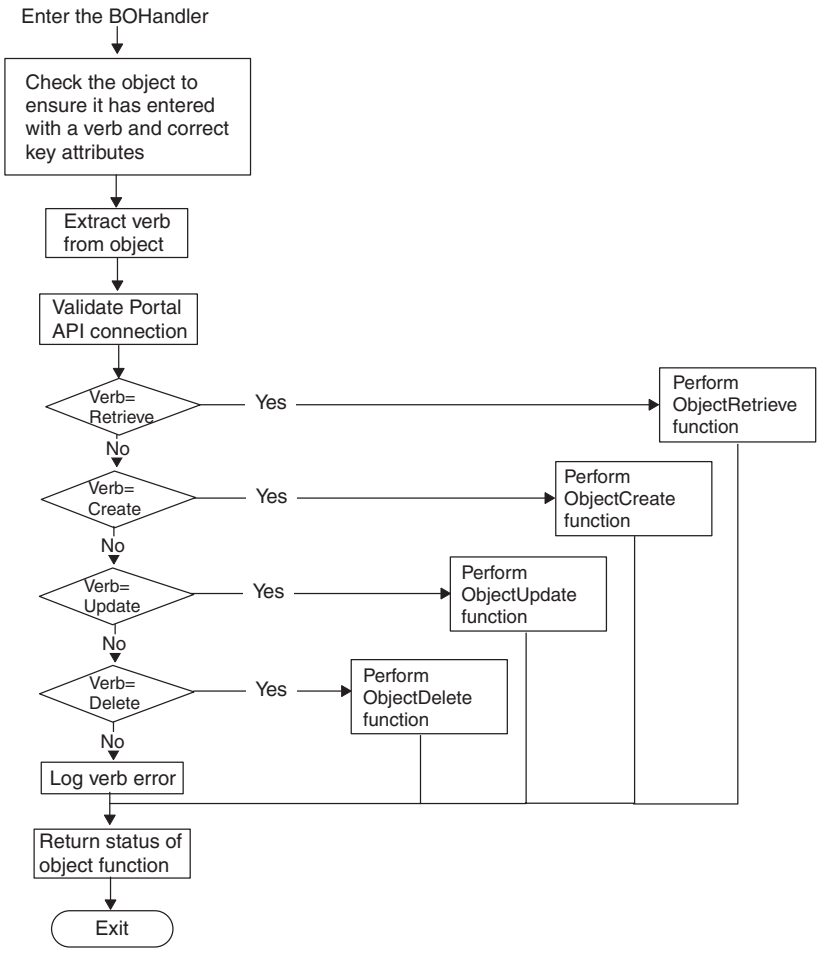


Figure 2. High-level view of business object processing

Retrieve verb processing

The business object handler Retrieve method retrieves an object from Portal Infranet and populates a WebSphere Business Integration Server Express 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 Server Express Adapter business object. Since the `flist` has a one-to-one correspondence to the Business Integration Server Express Adapter business object, the `flist` is traversed to retrieve all the attributes of the WebSphere Business Integration Server Express 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 InterChange Server Express.

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.

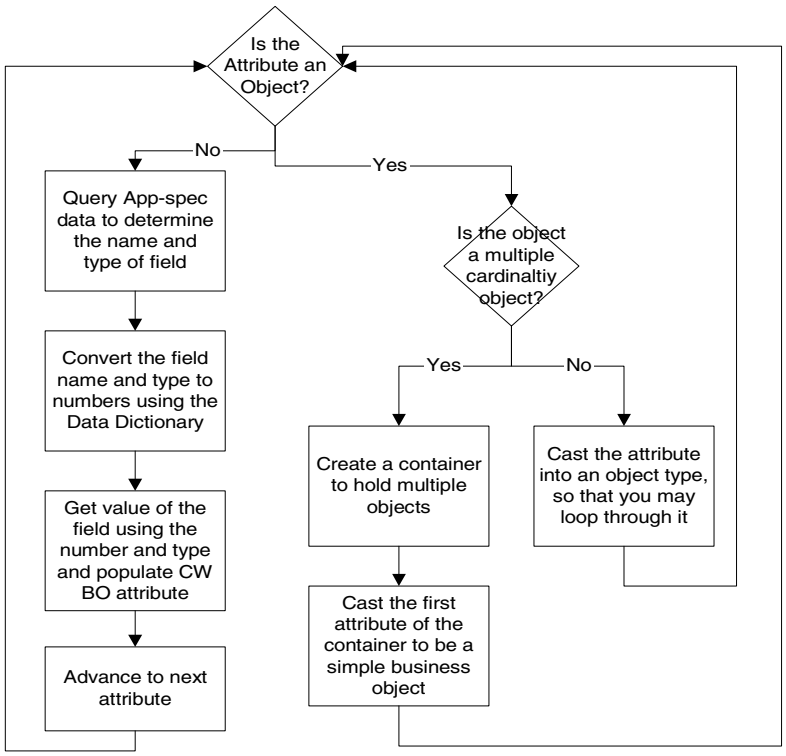


Figure 3. Flowchart for retrieve verb processing

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 Server Express 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 Business Integration Server Express 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 Server Express 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 InterChange Server Express, and it works with two configuration files to identify Infranet events and write events to the WebSphere Business Integration Server Express 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, InterChange Server Express provides a configuration flat file that configures Infranet to call the WebSphere Business Integration Server Express 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 Server Express 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 Server Express 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 Server Express 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 Server Express Adapter application-specific business object, sets the verb to the action in the event record, and then sends the business object to InterChange Server Express.

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 InterChange Server Express.

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.

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 Server Express 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 Server Express Adapter event facilities module. The event facilities module is an executable that implements event notification. This module selects the Infranet events relevant for InterChange Server Express 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. All file pathnames are relative to the directory where the WebSphere Business Integration Server Express Adapter product is installed on your system, unless otherwise noted.

Compatibility

The adapter framework that an adapter uses must be compatible with the version of InterChange Server Express with which the adapter is communicating. Version 4.3 of the adapter for Portal Infranet is supported on the following versions of the adapter framework and with the following integration brokers:

Adapter framework: WebSphere Business Integration Server Express Adapter Framework

Integration brokers: InterChange Server Express

See the Release Notes for any exceptions.

Note: For instructions on installing InterChange Server Express and its prerequisites, see the *Installation Guide for WebSphere Business Integration Server Express*.

Prerequisite software

The connector requires the following prerequisite software:

- Java Portal Communications Module (PCM) Library `pcm.jar` and `pcmext.jar`.
- All usual third-party software required by WebSphere Business Integration Server Express Adapter, including the WebLogic Oracle JDBC driver and library

The event module requires the following prerequisite software and settings:

- An installed Infranet application with an Oracle database
- The default value of the enviornment variable `ORACLE_SID`must be set to the alais name for the Portal Infranet database.

If you are installing on a Windows system, the event module also requires the following prerequisite software:

- Portal Communications Module (PCM) Library `pcm.dll`
- Microsoft MFC and dependencies, which are delivered with the connector component of the WebSphere Business Integration Server Express Adapter for Portal Infranet: `MFC42.dll`. If the system does not already have this file in the path, this file must be located in a directory defined in the PATH system variable. Placing it in the *`ProductDir`*`\bin` directory should satisfy that requirement.

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` 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 the adapter, refer to the *Installation Guide for WebSphere Business Integration Server Express*, located in the WebSphere Business Integration Server Express Infocenter at the following site:

http://www.ibm.com/websphere/wbiserverexpress/infocenter

Configuring the adapter in an AIX/DB2 environment

To configure the adapter on an AIX system, follow these instructions in order:

As DB2 user

1. Log in to the AIX system with the DB2 user ID used for the instance for Portal Infranet, for example, `db2instl`.
2. Enter the following command:

   ```
   db2 catalog tcpip node local_DB2_hostname remote remote_DB2_hostname
   server DB2_instance_portnumber
   ```

   Example:

```
db2 catalog tcpip node db2host remote db2host server 50000
```

3. Enter the following command:

   ```
   db2 catalog database DB_name as DB_name at node DB2_hostname
   ```

   Example:

   ```
   db2 catalog database CWPortal as CWPortal at node db2hos
   ```

4. Log off.

As user "pin"

1. Log in to the AIX system as pin (create a pin user account if necessary).
2. Copy the file db2profile into user pin's home directory, and add the line ../db2profile to the .profile file to execute the shell script file db2profile before starting the connector.

   The file is located in the directory sqllib, in the home directory of the account that was used to install Portal Infranet for DB2, for example, /home/db2instl/sqllib.

   Copy the file database.bnd from $COSSWORKDS/connectors/Portal/dependencies/config_files/DB2 to user pin's home directory.
3. Enter the following command:

   ```
   db2 connect to DB_name user pin using pin
   ```

   Example:

   ```
   db2 connect to CWPortal user pin using pin
   ```

4. Enter the following command:

   ```
   dbw bind database.bnd
   ```

5. Enter the following command:

   ```
   db2 disconnect all
   ```

6. Add the following line to the .profile file of user pin:

   ```
   export DB2DBDFT=DB_name
   ```

   Example:

   ```
   export DB2DBDFT=CWPortal
   ```

7. Log off pin.

As user "cwadmin"

1. Log in to the AIX system as user cwadmin, which is used to start up the connector.
2. Execute the shell script file db2profile before starting the connector.

   The file is located in the directory sqllib, in the home directory of the account that was used to install Portal Infranet for DB2, for example, /home/db2instl/sqllib.

   Copy the file connector_manager_Portal from $*ProductDir*/connectors/Portal to $*ProductDir*/connectors/bin.
3. Change the directory to $*ProductDir*/connectors/Portal. In the file start_Portal.sh, change the following line:

   PORTAL_HOME=/opt/portal/6.1

   to an appropriate value:

   PORTAL_HOME=PortalInfranet_home_dir

   Example:

   /opt/portal/6.2

4. Log off cwadmin.

As user “pin”

1. Log in to the AIX system as pin.
2. Copy the Infranet files pcm.jar and pcmext.jar into $*ProductDir*/connectors/Portal/dependencies. These files are located in the $INFRANET/jars directory of the Infranet server.
3. Copy the file custom_opcode.h to $INFRANET/include.
4. Copy the files custom_opcode_mem_map to $INFRANET/lib.
5. Copy the .profile file into pin’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 them into the system by typing the following command at the command prompt:

   ```
   ../.profile
   ```
6. Copy the file fm_crossworlds.so to the $INFRANET/lib directory. This file contains the triggers for the events.

   Note: AIX is case-sensitive, so if the files are not found, verify that all directory and file names have the proper case.

   Make sure that the $LIBRARY_PATH variable contains the $INFRANET/lib path so that the system can recognize the .so files.
7. Make sure that the $LIBRARY_PATH variable contains the $INFRANET/lib path so that the system can recognize the .so files.
8. Copy the following files into the directory $CW_PORTAL_PATH, for example, /opt/portal/6.2/sys/cm:
   - crossworlds.cnf -- This file contains configuration information for the event module.
   - event_code.txt -- This file contains descriptions of Infranet events that the event module uses to generate entries in the WebSphere business integration system event table.
9. Place the pin_notify_cw file 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 and /event/status.

Configuring Portal Infranet application in an AIX/DB2 environment

If you are using DB2 as your database and AIX as your operation system, follow these instructions to configure the Portal Infranet application:

1. 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 protal
      ```
   c. Kill any running Infranet processes with this command:

      ```
      kill -9 PID
      ```

d. Restart Infranet with the following command:

   `start_all`

2. In the `$CM` directory, add the following line to the `pin.conf` file in the `fm_required` section:

   `-cm fm_module $INFRANET/lib/fm_crossworlds.so fm_cw_config -pin`

   Be sure to type the full directory path for `$INFRANET`.
3. Verify that Infranet is running by entering the following command:

   `ps -ef | grep portal`
4. Change the 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`*

   Example:

   `nap_cm_ptr ip roadrunner 11960`

   where `roadruner` is *`Infranet_cm_machine`* and `11960` is *`cm_port`*
5. Verify that the `pin.conf` file includes:
6. `nap login_type 1 nap login_name root.0.0.0.1 nap login_pw password`

   This identifies the login information for connection to Infranet. If there is no `pin.conf` file in the directory, copy one onto the directory.
7. Load configuration information into the Infranet application by entering the command:

   `load_pin_notify pin_notify_cw`

   The response should read successful. If another response is shown, check `pin_notify_cw`. This file contains the opcodes that Infranet calls when specific events occur. Note that `pin_notify_cw` should be located in the same directory as the `load_pin_notify` executable.
8. In the `$INFRANET_VAR/cm` directory, check the log file and verify that there is a core in the `$CM`, for example, `cm started at "Tue Jul 9 20:49:37 2002"` in `cm.pinlog`. Start up the Infranet Administrator.
9. To test the connector, enter or modify an account and check the event table `xworlds_events` for the proper event entry. Because this results in a dummy event, delete the event entry after testing is complete.

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 6.1.0 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.
   - event_code.txt. This file contains descriptions of Infranet events that the event module will use to generate entries in the WebSphere Business Integration Server Express 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_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 25.

Installing and configuring the event mechanism for a Windows environment

Infranet has an event mechanism that permits it to keep track of the actions that occur in the application. You must configure the event notification mechanism in Infranet before the connector can process event delivery. Follow these steps:

1. Copy the following WebSphere Business Integration Server Express Adapter event module files:
   - Copy `FmCw.dll` to `%INFRANET%\lib`. This DLL is the connector event facilities module.
   - Copy `event_code.txt` to `%INFRANET%\sys\cm`. This file describes Infranet events to the event module, which generates entries corresponding to the Infranet events in the WebSphere Business Integration Server Express Adapter event table.

     Note that `%INFRANET%` is in the Infranet home directory.
   - Copy `custom_opcode.h` to `%INFRANET%\include`.
   - Copy `custom_opcode_mem_map` to `%INFRANET%\lib`.
   - Copy the Infranet files `pcm.jar` and `pcmext.jar` into `$CROSSWORKDS/Connector/Portal/dependencies`. This file is located in the `infranet/jars` directory of the Infranet server.
2. In `%INFRANET%\sys\cm`, edit the Connection Manager `pin.conf` configuration file so that the Connection Manager can execute the connector event facilities DLL, `FmCw.dll`. Add the following line to this file:

   ```
   - cm fm_module ..\..\lib\FmCw.dll fm_cw_pol_config_func – pin --
   ops_fields_extension_file ..\..\lib\custom_opcode_mem_map
   ```
3. Place the `pin_notify_cw` file in the `%INFRANET%\sys\test` directory. This file specifies which opcodes Infranet will call when specific events are raised. If any opcodes 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`.
4. Update the `pin.conf` file in `%INFRANET%\sys\test` to be something like the following:

   ```
   - nap cm ptr ip <Infranet_cm_machine> <port>
   ```

   For example:

   ```
   - nap cm ptr ip cwengtest 11960
   ```

   where `cwengtest` is <Infranet_cm_machine> and <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.
5. Copy the `crossworlds.cnf` environment configuration file to `%INFRANET%\sys\cm`. This file allows the event facilities module `FmCw.dll` to obtain information about the environment.

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

```
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 user`	Name of the user connecting to the Portal Infranet database.
`db password`	Password.
`crossworlds id`	POID representing the WebSphere Business Integration Server Express 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.

This file is interpreted as a hash table. The keywords are strings on the left of the equal sign, and the values are the strings on the right.

6. Create an environment variable named `CW_PORTAL_PATH` that specifies the path to the `crossworlds.cnf` file.
7. Restart Infranet services in this order:
   - Infranet Data Manager
   - Infranet Java Server
   - Infranet Connection Manager
8. Load the `pin_notify_cw` file into the Infranet application using the following syntax:

   ```
   load_pin_notify .\pin_notify_cw
   ```

   If errors occur, make sure that `\bin` directory is in the PATH so that the executable can be located. For error messages, check the `pinlog.log` in the execution directory.

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 Express which is accessed from System Manager. For detailed configuration information, see Appendix B, "Using Connector Configurator Express," on page 79 or the *Connector Development Guide for Java*.

Standard connector properties

Standard configuration properties provide information that all connectors use. See Appendix A, "Standard configuration properties for connectors," on page 65 for documentation of these properties.

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: Ensure that every component of your installation (for example, all adapters, applications, and InterChange Server Express itself), are set to the same locale.

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
DriverClass	*Driver class name of the database*		Yes
DbName	*Oracle Listener name*		Yes
DbPassword	*Database password*		Yes
DbUser	*Database user name*	`pin`	Yes
InfDatabase	*String*	`0.0.0.1`	Yes
InfHost	*Host machine name and port*	`//CWENGTEST1:11960`	Yes
InfLogFile	*Name of log file*	`InfConnection.txt`	Yes

Table 3. Connector-specific configuration properties (continued)

Name	Possible values	Default value	Required
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
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

DriverClass
Driver class name of the database.

DbName
Oracle Listener name for the Infranet database.

DbPassword
Database password.

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 22.

The IBM WebSphere Business Integration Server Express 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.

Note: Each additional instance of any adapter supplied with WebSphere Business Integration Server Express will be treated as a separate adapter by the function that limits the total number of adapters that can be deployed.

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 Express 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 Express. 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 24.
3. Create a startup script shortcut.
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.

Starting the connector

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

ProductDir\connectors*connName*

where *connName* identifies the connector. The name of the startup script depends on the operating-system platform, as Table 4 shows.

Table 4. Startup scripts for a connector

Operating system	Startup script
Windows	`start_connName.bat`

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

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

Select **Programs>IBM WebSphere Business Integration Express>Adapters>Connectors>*your_connector_name***.

By default, the program name is "IBM WebSphere Business Integration Express". 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 ICS [-cconfigFile ]`

  where *connName* is the name of the connector.
- From System Monitor

  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 the *System Administration Guide*.

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.
- From System Monitor

  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 31
- "Guidelines for defining business objects" on page 34
- "Business object application-specific information" on page 34

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 Server Express 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.

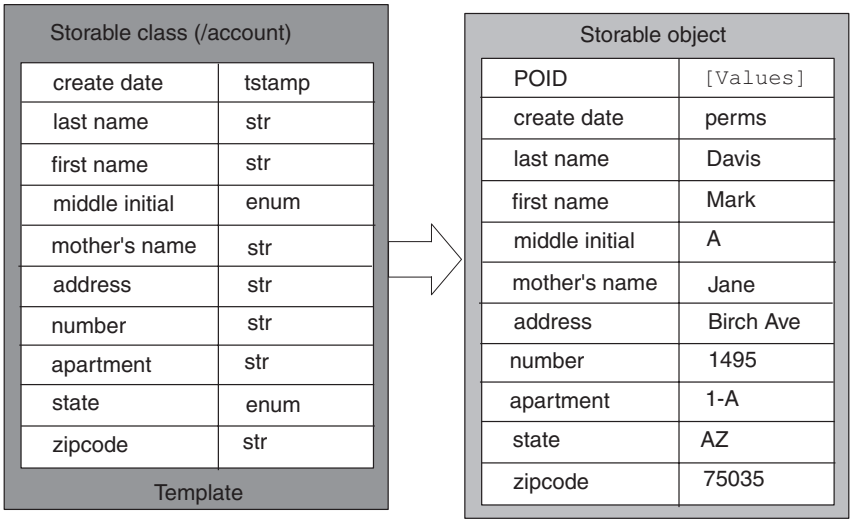


Figure 4. Infranet storable class and storable object

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.

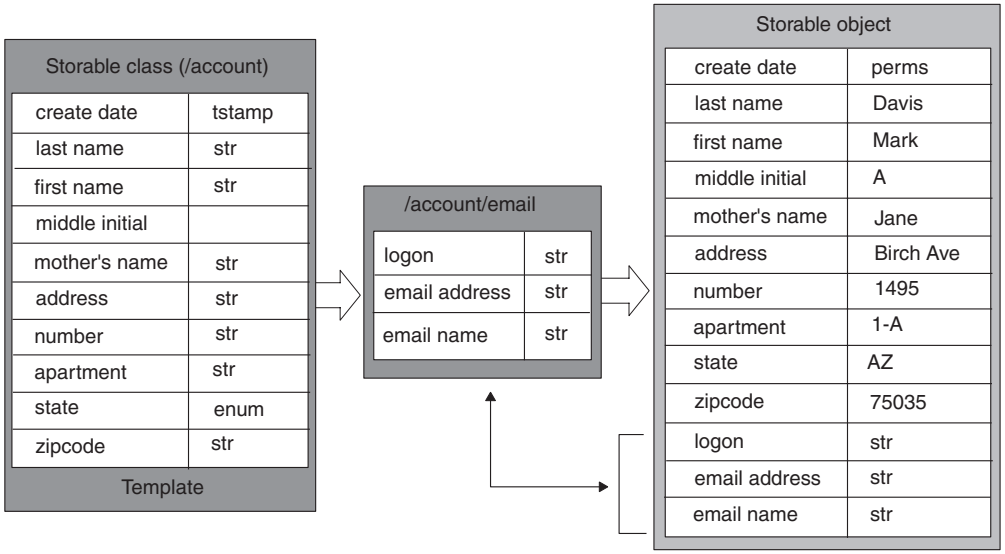


Figure 5. Extending the /account storable class

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.

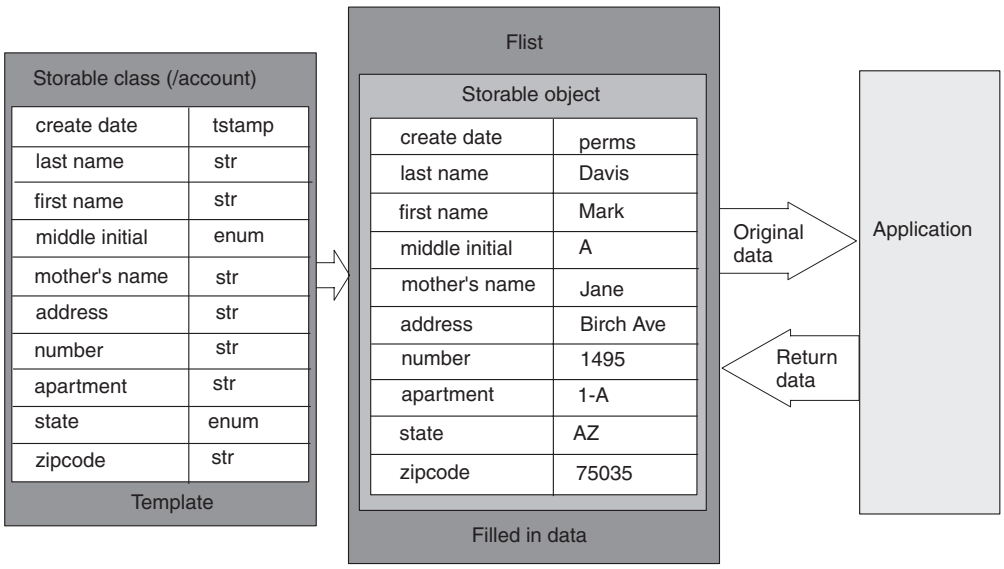


Figure 6. Storable object and flist

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 Server Express 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 Business Integration Server Express Adapter business objects

Infranet has the following container types:

- Storable class
- Array
- Substruct

You must define a Business Integration Server Express 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 Server Express 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 Business Integration Server Express Adapter business object and an Infranet flists might correspond. See the Portal Infranet documentation for information on Infranet flists.

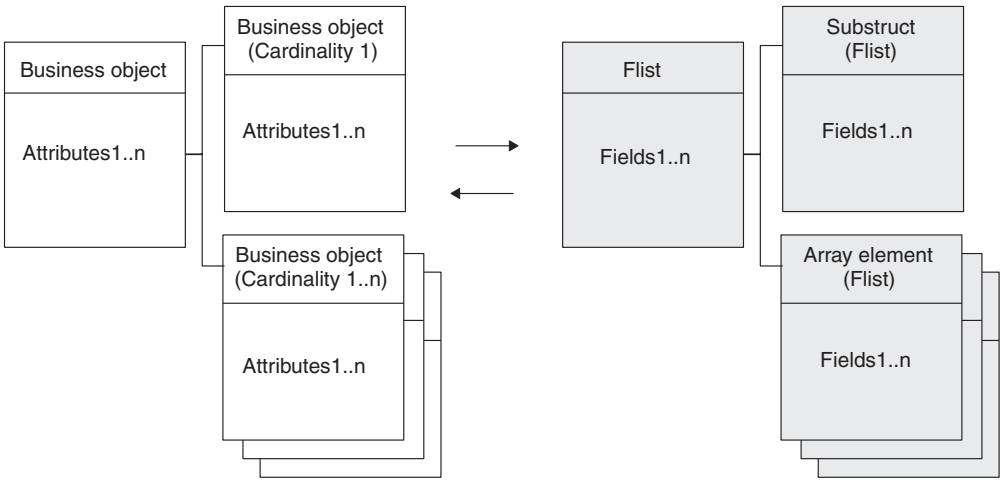


Figure 7. Structure business objects and flists

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.

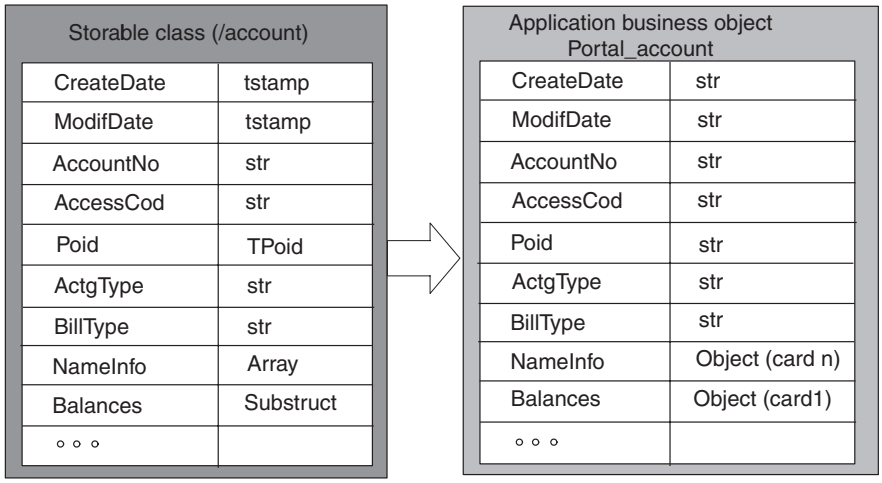


Figure 8. Coordinating of a storable class to a Business Integration Server Express Adapter business object

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.

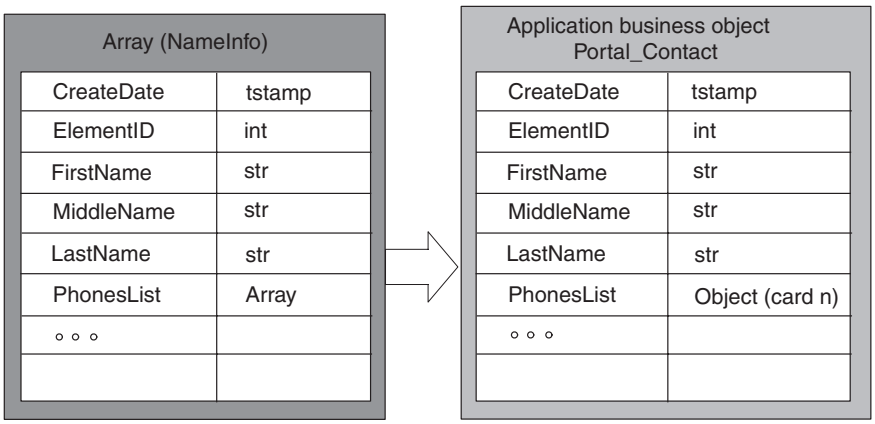


Figure 9. Corresponding of an flist array to a child business object

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 41.

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 18 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 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 41 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 41 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 35for instructions.

Utility business object definitions do not become business object instances that are sent through InterChange Server Express. 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 42.

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.

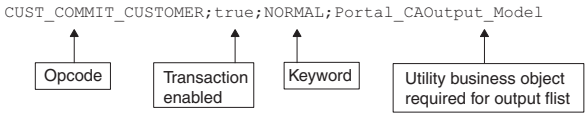
  ```
  [< bus_object_attributename>]([:< bus_object_attributename>])*
  ```

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 Server Express Adapter system. To enable the connector to obtain the new ID, Business Object Designer Express 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 InterChange Server Express. The utility business object is shown in Figure 11.

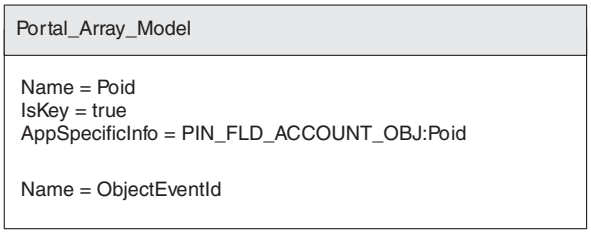


Figure 11. Portal_CAOutput_Model utility business object definition

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

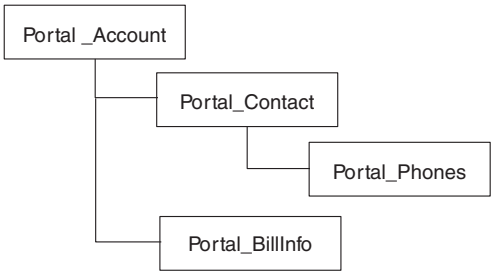


Figure 12. Diagram of the Portal_Account hierarchical business object

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.

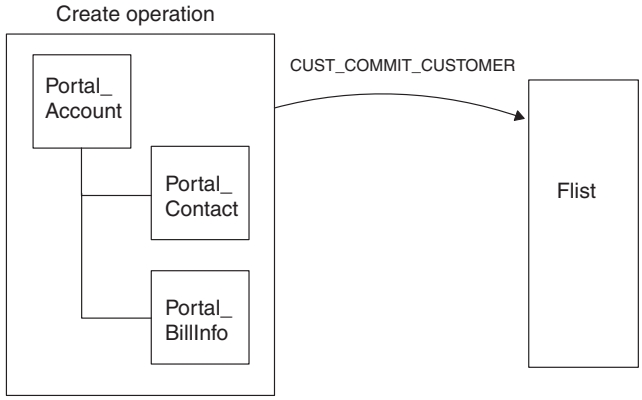


Figure 13. Flist for create operation for Portal_Account hierarchical business object

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.

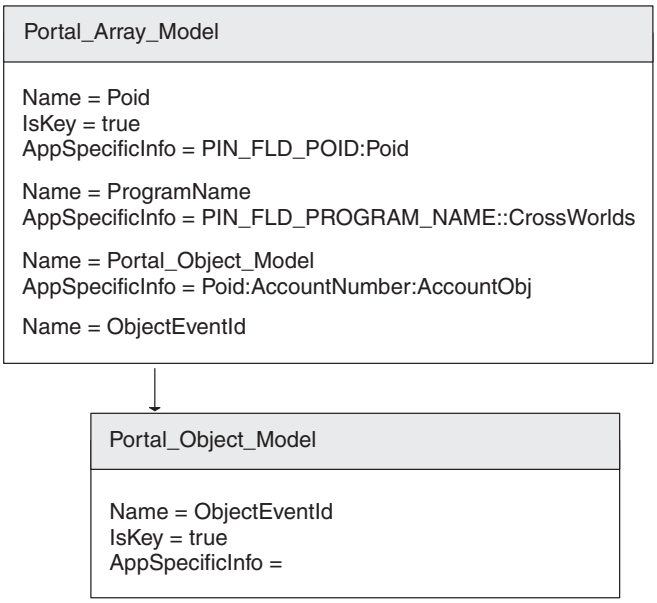


Figure 14. Portal_Array_Model utility business object definition

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.

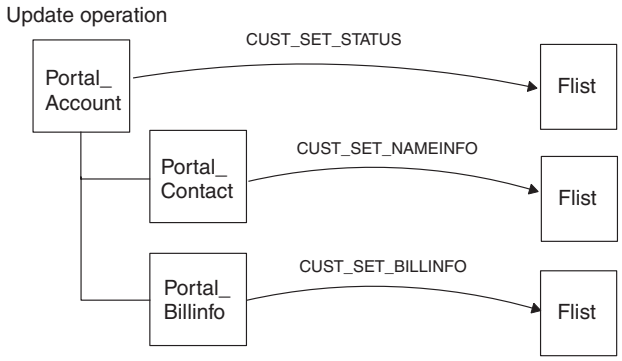


Figure 15. Flists for update operation for Portal_Account hierarchical business object

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 IBM WebSphere Business Integration Server Express Installer. Follow the instructions in the *Installation Guide for WebSphere Business Integration Server Express*. 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. All IBM WebSphere Business Integration Server Express 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 Express.
3. Follow a six-step process in Business Object Designer Express 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 Express. Business Object Designer Express 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 54 illustrates the window in Business Object Designer Express 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 54.

Using PortalODA in Business Object Designer Express

This section describes how to use PortalODA in Business Object Designer Express 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 Express, see the *Business Object Development Guide*.

After you start an ODA, you must start Business Object Designer Express to configure and run it. There are six steps in Business Object Designer Express to generate or convert a business object definition using an ODA. Business Object Designer Express 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 Express.
2. From the File menu, select the New Using ODA... submenu.

   Business Object Designer Express displays the first window in the wizard, named Select Agent. Figure 16 on page 54 illustrates this window.

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

1. "Select the ODA" on page 54
2. "Configure initialization properties" on page 54
3. "Generating definitions" on page 58 and, optionally, "Providing additional information" on page 59
4. "Saving definitions" on page 61

Select the ODA

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

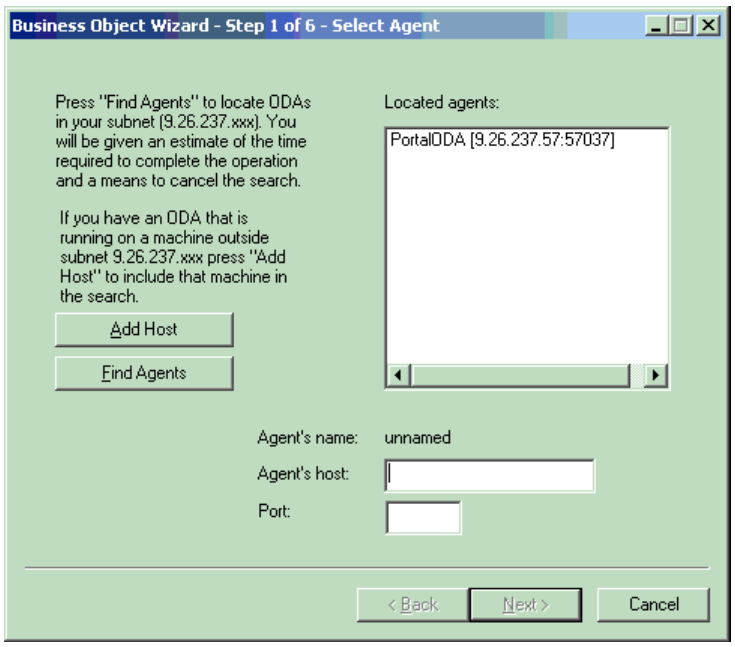


Figure 16. Selecting the ODA

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 Express does not locate your desired ODA, check the setup of the ODA.
2. Select the desired ODA from the displayed list.

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

Configure initialization properties

The first time Business Object Designer Express communicates with PortalODA, it prompts you to enter a set of initialization properties as shown in Figure 17. 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*.

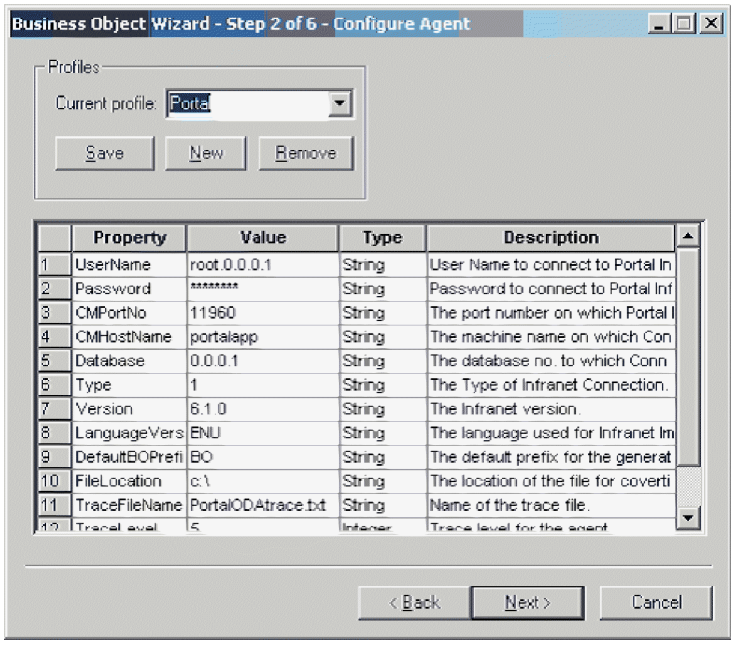


Figure 17. Configuring agent initialization properties

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 Express prompts you for business object properties. For more information, see “Providing additional information” on page 59
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 Express.

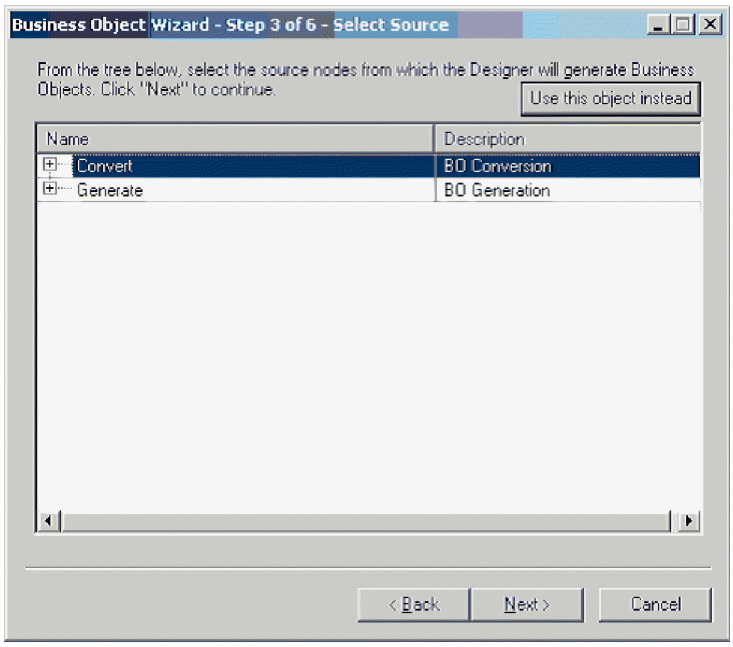


Figure 18. Tree giving two options for BO conversion and BO generation

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). 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 35 for details of various properties in application-specific information of an attribute.

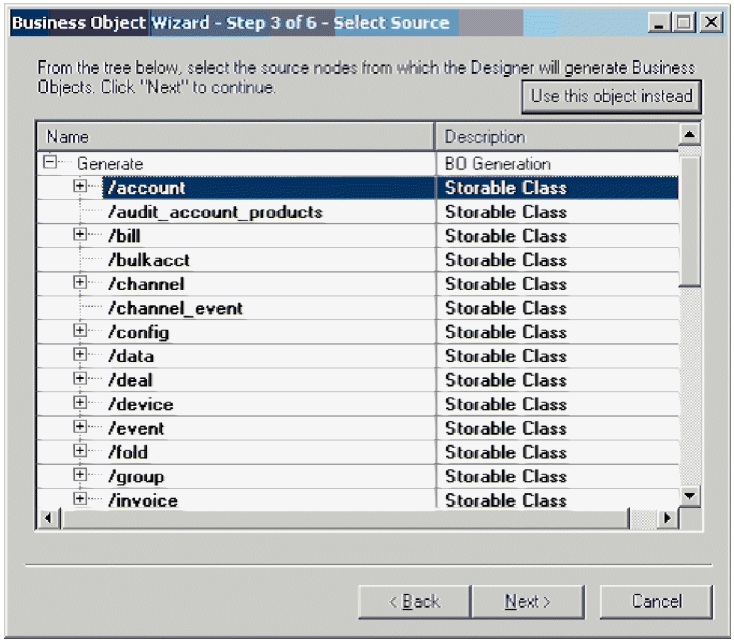


Figure 19. Screen showing the storable classes

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 Express displays the following confirmation screen (see Figure 20).

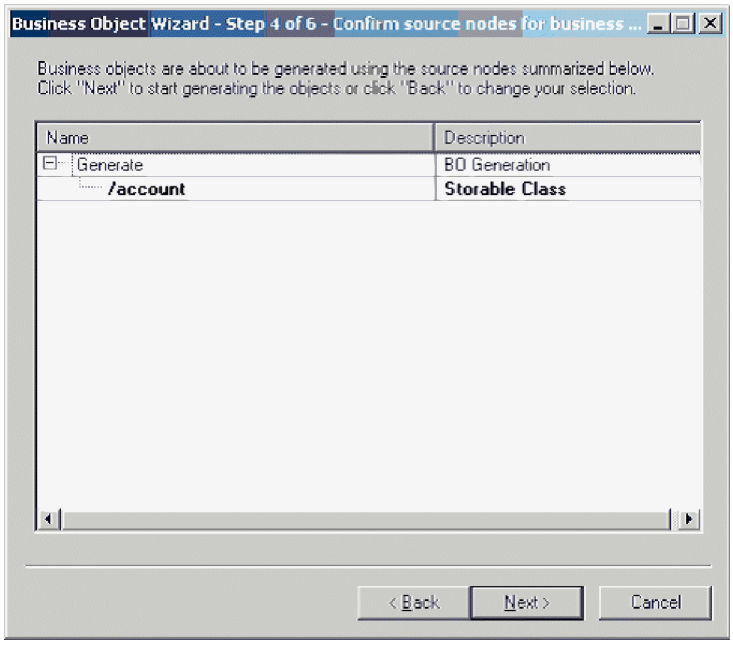


Figure 20. Confirming your selection

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 Express is generating the definitions.

Figure 21 illustrates this dialog box.

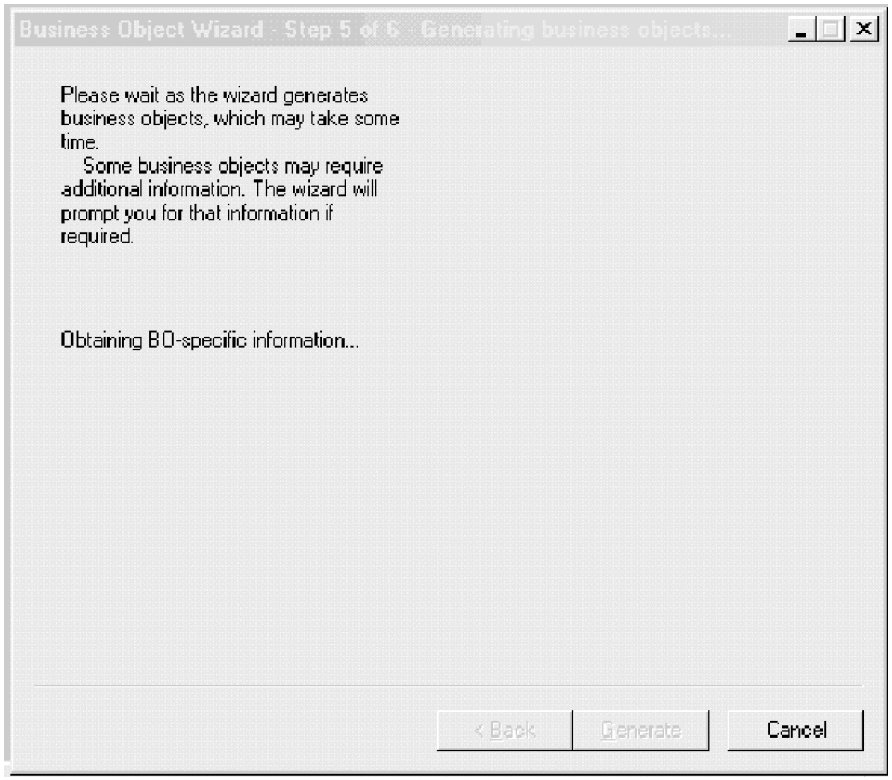


Figure 21. Generating definitions

Providing additional information

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

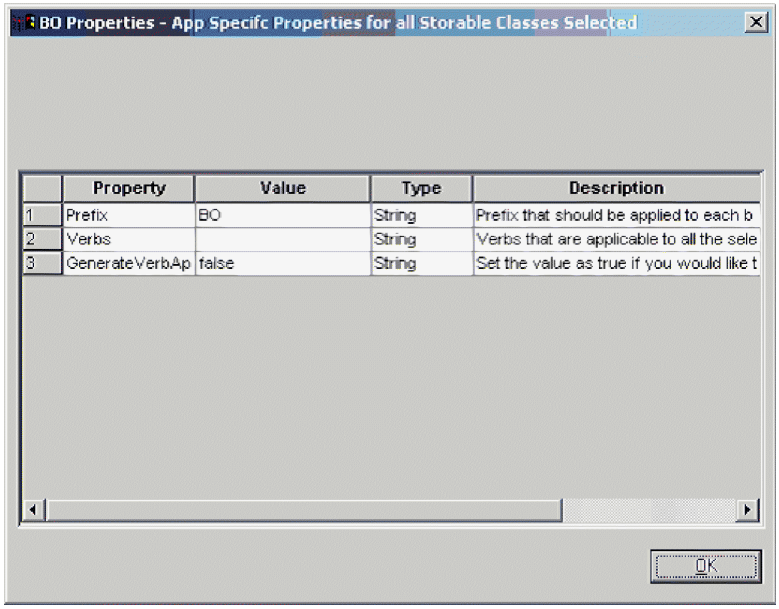


Figure 22. Application-specific properties for storable classes

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), 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.

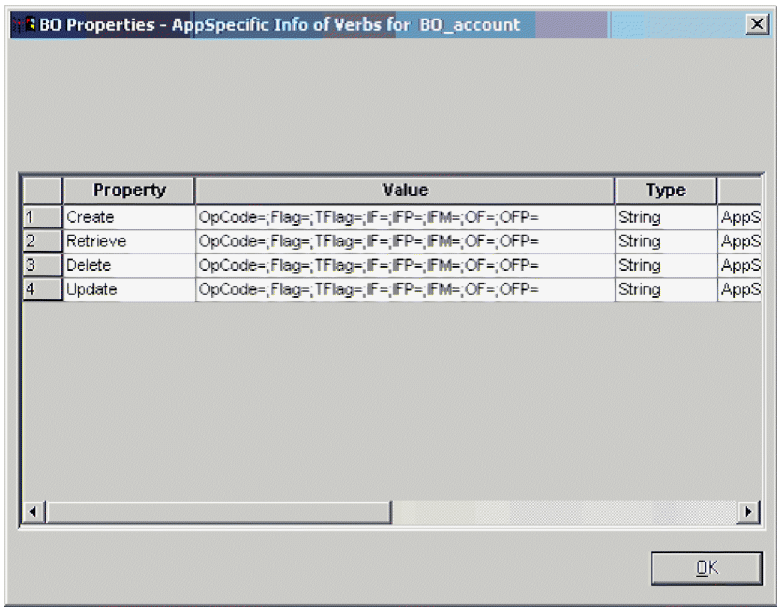


Figure 23. Application-specific information for verbs

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 Express 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 Express. For more information, and to make further modifications, see the *Business Object Development Guide*.

Figure 24 illustrates this dialog box.

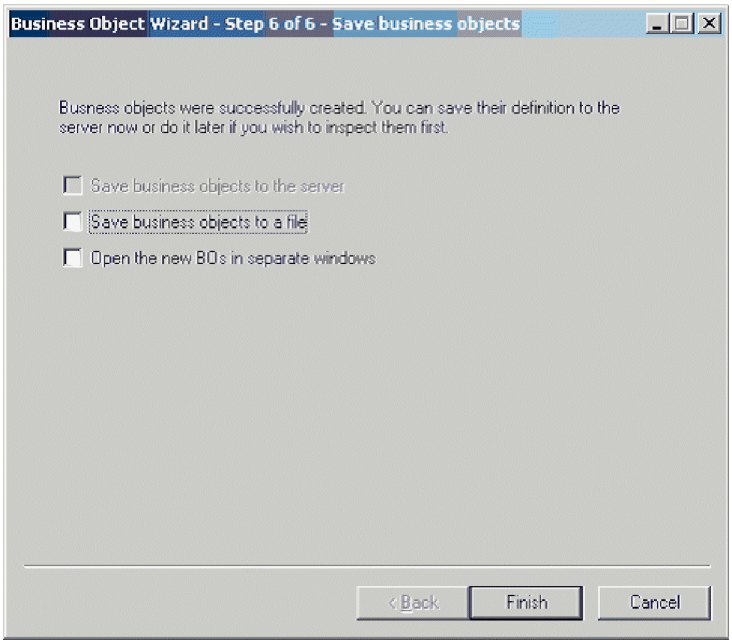


Figure 24. Saving the business object definition

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)
- 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 27.

This section describes:

- "Business-object-level properties" on page 62
- "Attribute properties" on page 63
- "Verbs" on page 64

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 61). 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 Express.

Note: Alternatively, because InterChange Server Express is the integration broker, you can use the repos_copy command to load the definition into the repository.

Appendix A. Standard configuration properties for connectors

This appendix describes the standard configuration properties for the connector component of the adapters in WebSphere Business Integration Server Express, running on WebSphere InterChange Server Express.

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

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

Configuring standard connector properties

Adapter connectors have two types of configuration properties:

- Standard configuration properties
- Connector-specific configuration properties

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

Using Connector Configurator Express

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

Setting and updating property values

The default length of a property field is 255 characters.

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

1. Default
2. Repository
3. Local configuration file
4. Command line

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

- **Dynamic**
  The change takes effect immediately after it is saved in System Manager.
- **Component restart**
  The change takes effect only after the connector is stopped and then restarted in System Manager. You do not need to stop and restart the application-specific component or the integration broker.

- **Server restart**
  The change takes effect only after you stop and restart the application-specific component and the integration broker.
- **Agent restart**
  The change takes effect only after you stop and restart the application-specific component.

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

Summary of standard properties

Table 11 provides a quick reference to the standard connector configuration properties. Not all the connectors make use of all these properties, and property settings may differ from integration broker to integration broker, as standard property dependencies are based on `RepositoryDirectory`.

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

Table 11. Summary of standard configuration properties

Property name	Possible values	Default value	Update method	Notes
AdminInQueue	Valid JMS queue name	*CONNECTORNAME* /ADMININQUEUE	Component restart	Delivery Transport is JMS
AdminOutQueue	Valid JMS queue name	*CONNECTORNAME*/ADMINOUTQUEUE	Component restart	Delivery Transport is JMS
AgentConnections	1-4	1	Component restart	Delivery Transport is IDL
AgentTraceLevel	0-5	0	Dynamic	
ApplicationName	Application name	Value specified for the connector application name	Component restart	
BrokerType	ICS	ICS		
CharacterEncoding	ascii7, ascii8, SJIS, Cp949, GBK, Big5, Cp297, Cp273, Cp280, Cp284, Cp037, Cp437 **Note:** This is a subset of supported values.	ascii7	Component restart	
ConcurrentEventTriggeredFlows	1 to 32,767	1	Component restart	Repository directory is <REMOTE>
ContainerManagedEvents	No value or JMS	No value	Component restart	Delivery Transport is JMS
ControllerStoreAndForwardMode	true or false	truetrue	Dynamic	Repository directory is <REMOTE>

Table 11. Summary of standard configuration properties (continued)

Property name	Possible values	Default value	Update method	Notes
ControllerTraceLevel	0-5	0	Dynamic	Repository directory is <REMOTE>
DeliveryQueue		*CONNECTORNAME*/DELIVERYQUEUE	Component restart	JMS transport only
DeliveryTransport	IDL or JMS	IDL	Component restart	
DuplicateEventElimination	true or false	false	Component restart	JMS transport only: Container Managed Events must be <NONE>
EnableOidForFlowMonitoring	true or false	false	Component restart	
FaultQueue		*CONNECTORNAME*/FAULTQUEUE	Component restart	JMS transport only
jms.FactoryClassName	CxCommon.Messaging.jms .IBMMQSeriesFactory or any Java class name	CxCommon.Messaging. jms.IBMMQSeriesFactory	Component restart	JMS transport only
jms.MessageBrokerName	crossworlds.queue. manager	crossworlds.queue.manager	Component restart	JMS transport only
jms.NumConcurrentRequests	Positive integer	10	Component restart	JMS transport only
jms.Password	Any valid password		Component restart	JMS transport only
jms.UserName	Any valid name		Component restart	JMS transport only
JvmMaxHeapSize	Heap size in megabytes	128m	Component restart	Repository directory is <REMOTE>
JvmMaxNativeStackSize	Size of stack in kilobytes	128k	Component restart	Repository directory is <REMOTE>
JvmMinHeapSize	Heap size in megabytes	1m	Component restart	Repository directory is <REMOTE>
Locale	en_US, ja_JP, ko_KR, zh_CN, zh_TW, fr_FR, de_DE, it_IT, es_ES, pt_BR **Note:** This is a subset of the supported locales.	en_US	Component restart	
LogAtInterchangeEnd	true or false	false	Component restart	
MaxEventCapacity	1-2147483647	2147483647	Dynamic	Repository Directory is <REMOTE>
MessageFileName	Path or filename	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	JMS transport only: DuplicateEvent Elimination must be true
OADAutoRestartAgent	`true` or `false`	`false`	Dynamic	Repository Directory is <REMOTE>
OADMaxNumRetry	A positive number	`1000`	Dynamic	Repository Directory is <REMOTE>
OADRetryTimeInterval	A positive number in minutes	`10`	Dynamic	Repository Directory is <REMOTE>
PollEndTime	`HH:MM` (HH is 0-23, MM is 0-59)	`HH:MM`	Component restart	
PollFrequency	A positive integer in milliseconds `no` (to disable polling) `key` (to poll only when the letter `p` is entered in the connector's Command Prompt window)	`10000`	Dynamic	
PollQuantity	`1-500`	`1`	Agent restart	JMS transport only: Container Managed Events is specified
PollStartTime	`HH:MM`(HH is 0-23, MM is 0-59)	`HH:MM`	Component restart	
RepositoryDirectory	Location of metadata repository		Agent restart	Set to <REMOTE>
RequestQueue	Valid JMS queue name	*CONNECTORNAME*/REQUESTQUEUE	Component restart	Delivery Transport is JMS
ResponseQueue	Valid JMS queue name	*CONNECTORNAME*/RESPONSEQUEUE	Component restart	Delivery Transport is JMS:
RestartRetryCount	`0-99`	`3`	Dynamic	
RestartRetryInterval	A sensible positive value in minutes: 1 - 2147483547	`1`	Dynamic	
SourceQueue	Valid JMS queue name	*CONNECTORNAME*/SOURCEQUEUE	Agent restart	Only if Delivery Transport is JMS and Container Managed Events is specified
SynchronousRequestQueue	Valid JMS queue name	*CONNECTORNAME*/ SYNCHRONOUSREQUESTQUEUE	Component restart	Delivery Transport is JMS

Table 11. Summary of standard configuration properties (continued)

Property name	Possible values	Default value	Update method	Notes
SynchronousRequestTimeout	0 - any number (millisecs)	0	Component restart	Delivery Transport is JMS
SynchronousResponseQueue	Valid JMS queue name	*CONNECTORNAME*/SYNCHRONOUSRESPONSEQUEUE	Component restart	Delivery Transport is JMS
WireFormat	CwBO	CwBO	Agent restart	

Standard configuration properties

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

AdminInQueue

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

The default value is CONNECTORNAME/ADMININQUEUE.

AdminOutQueue

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

The default value is CONNECTORNAME/ADMINOUTQUEUE.

AgentConnections

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

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

AgentTraceLevel

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

ApplicationName

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

BrokerType

Identifies the integration broker that you are using, which is ICS.

CharacterEncoding

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

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

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

ConcurrentEventTriggeredFlows

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

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

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

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

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

ContainerManagedEvents

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

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

The default value is No value.

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

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

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

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

ControllerStoreAndForwardMode

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

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

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

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

The default is true.

ControllerTraceLevel

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

DeliveryQueue

Applicable only if DeliveryTransport is JMS.

The queue that is used by the connector to send business objects to the WebSphere InterChange Server Express.

The default value is CONNECTORNAME/DELIVERYQUEUE.

DeliveryTransport

Specifies the transport mechanism for the delivery of events. Possible values are IDL for CORBA IIOP or JMS for Java Messaging Service. The default is IDL.

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

JMS

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

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

Important: There may be a memory limitation if you use the JMS transport mechanism for a connector running on WebSphere InterChange Server Express.

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.

DuplicateEventElimination

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

This property can also be set to `false`.

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

EnableOidForFlowMonitoring

When you set this property to `true`, the adapter framework will mark the incoming **ObjectEventId** as a foreign key for the purpose of flow monitoring.

The default is `false`.

FaultQueue

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

The default value is `CONNECTORNAME/FAULTQUEUE`.

JvmMaxHeapSize

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

The default value is 128m.

JvmMaxNativeStackSize

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

The default value is 128k.

JvmMinHeapSize

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

The default value is 1m.

jms.FactoryClassName

Specifies the class name to instantiate for a JMS provider. You *must* set this connector property when you choose `JMS` as your delivery transport mechanism (DeliveryTransport).

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

jms.MessageBrokerName

Specifies the broker name to use for the JMS provider. You *must* set this connector property when you choose JMS as your delivery transport mechanism (see DeliveryTransport).

The default is crossworlds.queue.manager.

jms.NumConcurrentRequests

Specifies the maximum number of concurrent service call requests that can be sent to a connector at the same time. Once that maximum is reached, new service calls block and wait for another request to complete before proceeding.

The default value is 10.

jms.Password

Specifies the password for the JMS provider. A value for this property is optional.

There is no default.

jms.UserName

Specifies the user name for the JMS provider. A value for this property is optional.

There is no default.

Locale

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

A locale name has the following format:

ll_TT.codeset

where:

ll	a two-character language code (usually in lower case)
TT	a two-letter country or territory code (usually in upper case)
codeset	the name of the associated character code set; this portion of the name is often optional.

By default, only a subset of supported locales appears in the drop list. To add other supported values to the drop list, you must manually modify the \Data\Std\stdConnProps.xml file in the product directory. For more information, see the appendix on using Connector Configurator Express in this guide.

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

http://www.ibm.com/software/websphere/wbiadapters/infocenter, or
http://www.ibm.com/websphere/integration/wicserver/infocenter

LogAtInterchangeEnd

Specifies whether to log errors to the integration broker's log destination. Logging to the broker's log destination also turns on e-mail notification, which generates e-mail messages for the MESSAGE_RECIPIENT specified in the InterchangeSystem.cfg file when errors or fatal errors occur.

For example, when a connector loses its connection to its application, if LogAtInterChangeEnd is set to true, an e-mail message is sent to the specified message recipient. The default is false.

MaxEventCapacity

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

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

MessageFileName

The name of the connector message file. The standard location for the message file is \connectors\messages. Specify the message filename in an absolute path if the message file is not located in the standard location.

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

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

MonitorQueue

The logical queue that the connector uses to monitor duplicate events. It is used only if the DeliveryTransport property value is JMS and DuplicateEventElimination is set to TRUE.

The default value is CONNECTORNAME/MONITORQUEUE

OADAutoRestartAgent

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

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

The default value is false.

OADMaxNumRetry

Specifies the maximum number of times that the MQ-triggered OAD automatically attempts to restart the connector after an abnormal shutdown. The OADAutoRestartAgent property must be set to true for this property to take effect.

The default value is 1000.

OADRetryTimeInterval

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

The default is 10.

PollEndTime

Time to stop polling the event queue. The format is HH:MM, where *HH* represents 0-23 hours, and *MM* represents 0-59 seconds.

You must provide a valid value for this property. The default value is HH:MM, but must be changed.

PollFrequency

The amount of time between polling actions. Set PollFrequency to one of the following values:

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

The default is 10000.

Important: Some connectors have restrictions on the use of this property. To determine whether a specific connector does, see the installing and configuring chapter of its adapter guide.

PollQuantity

Designates the number of items from the application that the connector should poll for. If the adapter has a connector-specific property for setting the poll quantity, the value set in the connector-specific property will override the standard property value.

PollStartTime

The time to start polling the event queue. The format is *HH:MM*, where *HH* represents 0-23 hours, and *MM* represents 0-59 seconds.

You must provide a valid value for this property. The default value is HH:MM, but must be changed.

RequestQueue

The queue that is used by WebSphere InterChange Server Express to send business objects to the connector.

The default value is CONNECTOR/REQUESTQUEUE.

RepositoryDirectory

The location of the repository from which the connector reads the XML schema documents that store the meta-data for business object definitions.

This value must be set to <REMOTE> because the connector obtains this information from the InterChange Server Express repository.

ResponseQueue

Applicable only if `DeliveryTransport` is JMS.

Designates the JMS response queue, which delivers a response message from the connector framework to the integration broker. WebSphere InterChange Server Express sends the request and waits for a response message in the JMS response queue.

RestartRetryCount

Specifies the number of times the connector attempts to restart itself. When used for a parallel connector, specifies the number of times the master connector application-specific component attempts to restart the slave connector application-specific component.

The default is 3.

RestartRetryInterval

Specifies the interval in minutes at which the connector attempts to restart itself. When used for a parallel connector, specifies the interval at which the master connector application-specific component attempts to restart the slave connector application-specific component. Possible values ranges from 1 to 2147483647.

The default is 1.

SourceQueue

Applicable only if `DeliveryTransport` is JMS and ContainerManagedEvents is specified.

Designates the JMS source queue for the connector framework in support of guaranteed event delivery for JMS-enabled connectors that use a JMS event store. For further information, see "ContainerManagedEvents" on page 70.

The default value is `CONNECTOR/SOURCEQUEUE`.

SynchronousRequestQueue

Applicable only if `DeliveryTransport` is JMS.

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

The default is `CONNECTORNAME/SYNCHRONOUSREQUESTQUEUE`

SynchronousResponseQueue

Applicable only if `DeliveryTransport` is JMS.

Delivers response messages sent in reply to a synchronous request from the broker to the connector framework. This queue is necessary only if the connector uses synchronous execution.

The default is `CONNECTORNAME/SYNCHRONOUSRESPONSEQUEUE`

SynchronousRequestTimeout

Applicable only if `DeliveryTransport` is JMS.

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

The default value is 0.

WireFormat

This is the message format on the transport. The setting is`CwBO`.

Appendix B. Using Connector Configurator Express

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

The topics covered in this appendix are:

Overview of Connector Configurator Express

Connector Configurator Express allows you to configure the connector component of your adapter for use with WebSphere InterChange Server Express.

You use Connector Configurator Express 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 maps for use with collaborations as well as specify messaging, logging and tracing, and data handler parameters, as required.

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 Express 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 Express.

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 Express will show the properties available for your particular configuration.

For **connector-specific properties**, however, you need first to define the properties and then set their values. You do this by creating a connector-specific property template for your particular adapter. There may already be a template set up in your system, in which case, you simply use that. If not, follow the steps in "Creating a new template" on page 81 to set up a new one.

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

Starting Connector Configurator Express

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

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

Running Configurator Express in stand-alone mode

You can run Connector Configurator Express independently and work with connector configuration files, irrespective of your broker.

To do so:

- From **Start>Programs**, click **IBM WebSphere Business Integration Server Express> Toolset Express>Development>Connector Configurator Express**.
- Select **File>New>Configuration File**.

You may choose to run Connector Configurator Express 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 85.)

Running Configurator Express from System Manager

You can run Connector Configurator Express from System Manager.

To run Connector Configurator Express:

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 Express**. The Connector Configurator Express window opens and displays a **New Connector** dialog box.

To edit an existing configuration file:

1. In the System Manager window, select any of the configuration files listed in the Connector folder and right-click on it.
2. Click the Standard Properties tab to see which properties are included in this configuration file.

Creating a connector-specific property template

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

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

- To create a new template, see "Creating a new template" on page 81.
- To use an existing file, simply modify an existing template and save it under the new name.

Creating a new template

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

To create a template:

1. Click **File>New>Connector-Specific Property Template**.
2. The **Connector-Specific Property Template** dialog box appears, with the following fields:
   - **Template**, and **Name**

     Enter a unique name that identifies the connector, or type of connector, for which this template will be used. You will see this name again when you open the dialog box for creating a new configuration file from a template.
   - **Old Template**, and **Select the Existing Template to Modify**

     The names of all currently available templates are displayed in the Template Name display.
   - To see the connector-specific property definitions in any template, select that template's name in the **Template Name** display. A list of the property definitions contained in that template will appear in the **Template Preview** display. You can use an existing template whose property definitions are similar to those required by your connector as a starting point for your template.
3. Select a template from the **Template Name** display, enter that template name in the **Find Name** field (or highlight your selection in **Template Name**), and click **Next**.

If you do not see any template that displays the connector-specific properties used by your connector, you will need to create one.

Specifying general characteristics

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

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

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

Specifying values

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

1. Click the **Value** tab. The display panel for Value replaces the display panel for General.

2. Select the name of the property in the **Edit properties** display.
3. In the fields for **Max Length** and **Max Multiple Values**, make any changes. The changes will not be accepted unless you also open the **Property Value** dialog box for the property, described in the next step.
4. Right-click the box in the top left-hand corner of the value table and click **Add**. A **Property Value** dialog box appears. Depending on the property type, the dialog box allows you to enter either a value, or both a value and range. Enter the appropriate value or range, and click **OK**.
5. The **Value** panel refreshes to display any changes you made in **Max Length** and **Max Multiple Values**. It displays a table with three columns:

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

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

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

   After a value has been created and appears in the grid, it can be edited from within the table display. To make a change in an existing value in the table, select an entire row by clicking on the row number. Then right-click in the **Value** field and click **Edit Value**.

Setting dependencies

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

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

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

   == (equal to)

   != (not equal to)

   > (greater than)

   < (less than)

   >= (greater than or equal to)

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

Creating a new configuration file

You create a connector configuration file from a connector-specific template or by modifying an existing configuration file.

Creating a configuration file from a connector-specific template

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

1. Click **File>New>Connector Configuration**.
2. The **New Connector** dialog box appears, with the following fields:
   - **Name**

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

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

     The default broker is ICS. You cannot change this value.
   - **Select Connector-Specific Property Template**

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

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

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

Using an existing file

To use an existing file to configure a connector, you must open the file in Connector Configurator Express, revise the configuration, and then save the file as a configuration file (`*.cfg`).

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 InterChange Server Express repository file.
  Definitions used in a previous InterChange Server Express 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 Express, 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 Express, 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`)
   - InterChange Server Express Repository (`*.in`, `*.out`)

     Choose this option if a repository file was used to configure the connector. 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 Express.
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 Express window displays the configuration screen, with the current attributes and values.

Connector Configurator Express requires values for properties described in the following sections:

- "Setting standard connector properties"
- "Setting application-specific configuration properties" on page 86
- "Specifying supported business object definitions" on page 87
- "Associated maps" on page 88
- "Setting trace/log file values" on page 89

Note: For connectors that use JMS messaging, an additional category may display, for special configuration of data handlers that convert the data to business objects. For further information, see "Data handlers" on page 90.

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 Express displays a configuration screen with tabs for the categories of required configuration values.

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.
- You can configure Value fields.
- The **Update Method** displayed for each property indicates whether a component or agent restart is necessary to activate changed values.

Setting standard connector properties

To change the value of a standard property:

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

3. After entering all the values for the standard properties, you can do one of the following:
   - To discard the changes, preserve the original values, and exit Connector Configurator Express, click **File>Exit** (or close the window), and click **No** when prompted to save changes.
   - To enter values for other categories in Connector Configurator Express, select the tab for the category. The values you enter for **Standard Properties** (or any other category) are retained when you move to the next category. When you close the window, you are prompted to either save or discard the values that you entered in all the categories as a whole.
   - To save the revised values, click **File>Exit** (or close the window) and click **Yes** when prompted to save changes. Alternatively, click **Save>To File** from either the File menu or the toolbar.

Setting application-specific configuration properties

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

1. Right-click in the top left portion of the grid. A pop-up menu bar will appear. Click **Add** to add a property. To add a child property, right-click on the parent row number and click **Add child**.
2. Enter a value for the property or child property.
3. To encrypt a property, select the **Encrypt** box.
4. Choose to save or discard changes, as described for "Setting standard connector properties" on page 85.

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

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

Encryption for connector properties

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

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

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

Update method

Refer to the descriptions of update methods found in the *Standard configuration properties for connectors* appendix, under "Setting and updating property values" on page 65.

Connector properties are almost all static and the **Update Method** is `Component restart`. For changes to take effect, you must restart the connector after saving the revised connector configuration file.

Specifying supported business object definitions

Use the **Supported Business Objects** tab in Connector Configurator Express 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.

For you to specify a supported business object, the business objects and their maps must exist in the system. Business object definitions, including those for data handler meta-objects, and map definitions should be saved into Integration Component Library (ICL) projects. For more information on ICL projects, see the *User Guide for WebSphere Business Integration Server Express*.

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 chapter on business objects in this guide as well as the *Business Object Development Guide*.

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

Business object name

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

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

To delete a business object from the supported list:

1. To select a business object field, click the number to the left of the business object.
2. From the **Edit** menu of the Connector Configurator Express 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 Express 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.

Associated maps

Each connector supports a list of business object definitions and their associated maps that are currently active in InterChange Server Express. 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 Express window.
- **Associated Maps**

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

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

  Explicit binding is required only when more than one map exists for a particular supported business object. When InterChange Server Express 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 Express window, click **Save to Project**.
4. Deploy the project to InterChange Server Express.
5. Reboot the server for the changes to take effect.

Resources

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.

Setting trace/log file values

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

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. Adapters that make use of the guaranteed event delivery enable this tab.

See the descriptions under `ContainerManagedEvents` in the Standard Properties appendix for values to use for these properties.

Saving your configuration file

After you have created the configuration file and set its properties, you need to deploy it to the correct location for your connector. Save the configuration in an ICL project, and use System Manager to load the file into InterChange Server Express.

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

- From System Manager, as a file with a `*.con` extension in an Integration Component Library, or
- In a directory that you specify.
- In stand-alone mode, as a file with a `*.cfg` extension in a directory folder.

For details about using projects in System Manager, and for further information about deployment, see the *User Guide for IBM WebSphere Business Integration Server Express.*

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 Express in a globalized environment

Connector Configurator Express is globalized and can handle character conversion between the configuration file and the integration broker. Connector Configurator Express uses native encoding. When it writes to the configuration file, it uses UTF-8 encoding.

Connector Configurator Express 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>
```

Notices

IBM may not offer the products, services, or features discussed in this document in all countries. Consult your local IBM representative for information on the products and services currently available in your area. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any non-IBM product, program, or service.

IBM may have patents or pending patent applications covering subject matter described in this document. The furnishing of this document does not give you any license to these patents. You can send license inquiries, in writing, to:

IBM Director of Licensing
IBM Corporation
North Castle Drive
Armonk, NY 10504-1785
U.S.A.

The following paragraph does not apply to the United Kingdom or any other country where such provisions are inconsistent with local law:

INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Some states do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement may not apply to you.

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make improvements and/or changes in the product(s) and/or program(s) described in this publication at any time without notice.

Any references in this information to non-IBM Web sites are provided for convenience only and do not in any manner serve as an endorsement of those Web sites. The materials at those Web sites are not part of the materials for this IBM product and use of those Web sites is at your own risk.

IBM may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation to you.

Licensees of this program who wish to have information about it for the purpose of enabling: (i) the exchange of information between independently created programs and other programs (including this one) and (ii) the mutual use of the information which has been exchanged, should contact:

IBM Burlingame Laboratory Director
IBM Burlingame Laboratory
577 Airport Blvd., Suite 800

Burlingame, CA 94010
U.S.A

Such information may be available, subject to appropriate terms and conditions, including in some cases, payment of a fee.

The licensed program described in this document and all licensed material available for it are provided by IBM under terms of the IBM Customer Agreement, IBM International Program License Agreement, or any equivalent agreement between us.

Any performance data contained herein was determined in a controlled environment. Therefore, the results obtained in other operating environments may vary significantly. Some measurements may have been made on development-level systems and there is no guarantee that these measurements will be the same on generally available systems. Furthermore, some measurement may have been estimated through extrapolation. Actual results may vary. Users of this document should verify the applicable data for their specific environment.

Information concerning non-IBM products was obtained from the suppliers of those products, their published announcements or other publicly available sources. IBM has not necessarily tested those products and cannot confirm the accuracy of performance, compatibility or any other claims related to non-IBM products. Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products.

This information may contain examples of data and reports used in daily business operations. To illustrate them as completely as possible, the examples may include the names of individuals, companies, brands, and products. All of these names are fictitious and any similarity to the names and addresses used by an actual business enterprise is entirely coincidental.

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

Programming interface information

Programming interface information, if provided, is intended to help you create application software using this program.

General-use programming interfaces allow you to write application software that obtain the services of this program's tools.

However, this information may also contain diagnosis, modification, and tuning information. Diagnosis, modification and tuning information is provided to help you debug your application software.

Warning: Do not use this diagnosis, modification, and tuning information as a programming interface because it is subject to change.

Trademarks and service marks

The following terms are trademarks or registered trademarks of International Business Machines Corporation in the United States or other countries, or both:

IBM
the IBM logo
AIX
CrossWorlds
DB2
DB2 Universal Database
Domino
Lotus
Lotus Notes
MQIntegrator
MQSeries
Tivoli
WebSphere

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

MMX, Pentium, and ProShare are trademarks or registered trademarks of Intel Corporation in the United States, other countries, or both.

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

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

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

IBM WebSphere Business Integration Server Express V4.3 and WebSphere Business Integration Server Express Plus V4.3.

Printed in USA