IBM WebSphere Business Integration Adapters

Adapter for PeopleSoft User Guide

Version 2.3.x

Note!
Before using this information and the product it supports, read the information in Appendix C, "Notices," on page 111.

19December2003

This edition of this document applies to IBM WebSphere Integration Adapter for PeopleSoft, version 2.3, and to all subsequent releases and modifications until otherwise indicated in new editions.

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

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

Contents

About this document

The IBM(R) WebSphere(R) Business Integration Adapter portfolio supplies integration connectivity for leading e-business technologies, enterprise applications, legacy, and mainframe systems. The product set includes tools and templates for customizing, creating, and managing components for business process integration.

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

Audience

This document is for IBM consultants and customers. You should be familiar with PeopleSoft and WebSphere business integration system adapter development.

Related documents

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

This document contains many references to two other documents: the System Installation Guide for Windows or for UNIX and the Implementation Guide for WebSphere InterChange Server. If you choose to print this document, you may want to print these documents as well.

You can install documentation from the following sites:

- "For general adapter information; for using adapters with WebSphere message brokers (WebSphere MQ Integrator, WebSphere MQ Integrator Broker, WebSphere Business Integration Message Broker); and for using adapters with WebSphere Application Server:

  http://www.ibm.com/websphere/integration/wbiadapters/infocenter
- For using adapters with InterChange Server:

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

  http://www.ibm.com/websphere/integration/wbicollaborations/infocenter
- For more information about message brokers (WebSphere MQ Integrator Broker, WebSphere MQ Integrator, and WebSphere Business Integration Message Broker):

  http://www.ibm.com/software/integration/mqfamily/library/manualsa/.
- For more information about WebSphere Application Server:

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

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

Typographic conventions

This document uses the following conventions:

courier font	Indicates a literal value, such as a command name, file name, information that you type, or information that the system prints on the screen.
bold	Indicates a new term the first time that it appears.
italic, *italic*	Indicates a variable name or a cross-reference.
blue text	Blue text, which is visible only when you view the manual online, indicates a cross-reference hyperlink. Click any blue text 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.
< >	Angle brackets surround individual elements of a name to distinguish them from each other, as in <server_name><*connector_name*>tmp.log.

Naming conventions

In this document the following naming conventions are used:

- The connector component of the IBM WebSphere Business Integration Adapter for PeopleSoft is referred to simply as the connector.
- The "connector" refers to the combination of the Vision Connector Framework and a connector module.

New in this release

New in Release 2.3.x

The following are new in this release:

- Support for processing retrieve-by-content requests.
  This enables record retrieval by alternate-key values.
- The adapter now supports BigDecimal type data.
- The connector can now be configured to run multi-threaded when processing inbound objects. When the connector runs multi-threaded, one session instance is reserved for polling which is performed serially, single-threaded.
- PeopleTools 8.42 and 8.43 are supported.
- Additional parameters have been added to address limitations encountered when trying to insert effective-dated items using a call in the PeopleSoft API library.
- Delete requests are supported for outbound objects.
- Adapter installation information has been moved from this guide. See Chapter 2 for the new location of that information.
- Beginning with the 4.3 version, the adapter for PeopleSoft is no longer supported on Microsoft Windows NT.

New in Release 2.2.x

The following are new in this release:

- The Adapter for PeopleSoft is now supported on HP-UX
- A single ODA instance can be reused several times to connect to the same application server
  The connector terminates when the connection to the application server is lost or the session instance becomes invalid during a transaction. Configurable connector termination support has been added
- This connector adds a new effective-dating sequence attribute that allows data to be inserted based on the sequence number. AResponseTimeOut is now supported (PingCompInterface)

New in Release 2.1.x

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

The following features are new in this release:

- The PeopleSoft Adapter, which includes the PeopleSoft ODA, only fully supports version 8.18 and later of PeopleTools, as well as versions 8.4 and later.
- Because PeopleTools now supports Application Designer projects in XML, all previous versions of the CrossWorlds event notification objects must be updated or imported in the new format. See PeopleSoft support for instructions.

- The Business Object ASI has been extended to include the new PeopleSoft property setEditHistoryItems. When this value is set to true, the adapter can run in Correction mode, which allows changes to history records in PeopleSoft. Before this value was incorporated into the getHistoryItems property, but with releases of PeopleTools 8.4 and later, this has been extracted into two separate properties, getHistoryItems, which retrieves history items, and setEditHistoryItems, which allows editing. Both of these properties are supported in this version of the adapter for PeopleSoft. See the PeopleSoft documentation on Component Interfaces for details on the functionality of these properties.
- Internal PeopleSoft messages are now enabled in the adapter log files. If there are PeopleCode processing errors or warnings in the context of the current component interface, the connector retrieves the messages from the PSMessage Collection object and displays them. See PeopleSoft Component Interface documentation for a full explanation of the content of these messages.

New in Release 2.0.x

The connector delivered with IBM WebSphere Business Integration Adapter for PeopleSoft has been internationalized. For more information, see "Processing locale-dependent data" on page 9 and Appendix A, "Standard configuration properties for connectors."

New in Release 1.4.x

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

This adapter includes:

- An application-component specific to PeopleSoft
- PeopleSoftODA for PeopleTools Version 8.16 or later
- A sample business object, which is located in \connectors\PeopleSoft\Samples\
- IBM WebSphere Adapter Framework, which consists of:
  - Connector Framework
  - Development tools (including Business Object Designer and Connector Configurator)
  - APIs (including ODK, JCDK, and CDK)

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

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

New in Release 1.3.x

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

A new tool, PeopleSoftODA, has been developed for generating business object definitions for the connector. For more information, see Chapter 4, "Generating business object definitions using PeopleSoftODA," on page 53.

New in Release 1.2.x

This release of the document for connector version 1.2.x contains the following new or corrected information:

- Events that have a status of "In Status" (3) are reset to `ReadyToPoll` (0) by the connector during `Init()`.
- Connector Properties `GetHistoryItems` and `setInteractiveMode` have been removed. This functionality is now defined at the business-object level application-specific information in the form of name value pairs. The default for both is `true`.
- A new connector property `ConnectorID` is available to support Event Distribution.

New in Release 1.1.x

This release of the document for connector version 1.1.x contains the following new or corrected information:

- Event notification support: "Event-processing components" on page 3, "Processing application events" on page 6, "Installation operations required to process application events" on page 16, "Code for processing application events" on page 19, "Event and archive tables" on page 20.
- Four new connector-specific configuration properties: "ConvertToPrimitiveFloat" on page 23, "EventKeyDelimiter" on page 23, "SetLangCode" on page 24, and "ReconnectSessionOnGetFail" on page 24.

Chapter 1. Overview of the connector

This chapter describes the connector component of the IBM WebSphere Business Integration Adapter for PeopleSoft. The connector enables an integration broker to exchange business objects with PeopleSoft version 8 applications that use PeopleTools Version 8.14 or later.

Connectors consist of an application-specific component and the connector framework. The application-specific component contains code tailored to a particular application. The connector framework, whose code is common to all connectors, acts as an intermediary between the integration broker and the application-specific component. The connector framework provides the following services between the integration broker and the application-specific component:

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

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

For more information about the relationship of the integration broker to the connector, see the *IBM WebSphere InterChange Server System Administration Guide*, or the *Implementation Guide for MQ Integrator*.

Connector components

The IBM WebSphere Business Integration Adapter for PeopleSoft contains the connector and at least one PeopleSoft Business Component and Component Interface. Figure 1 shows the connector and its relationship to the PeopleSoft application.

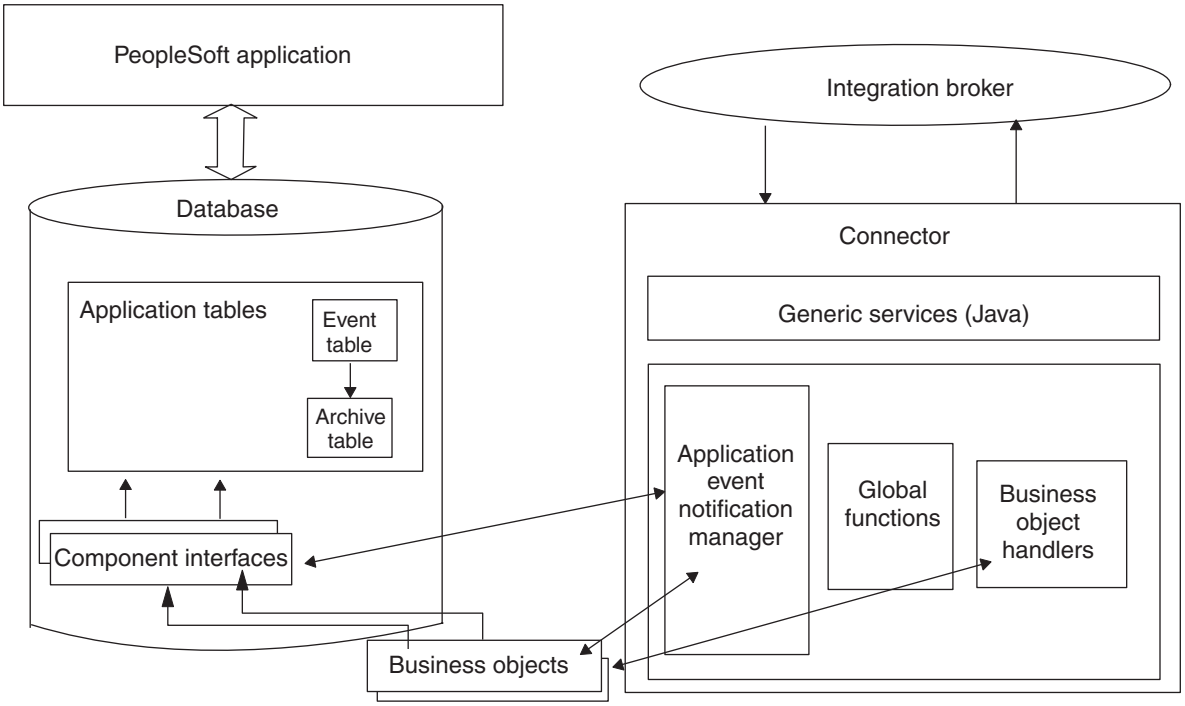

Figure 1. Connector architecture

Connector for PeopleSoft 8

The connector, which is written in Java, complies with IBM business integration system standards for adapters. It establishes a PeopleSoft session object, and uses the standard connector methods to process components. The connector supports PeopleSoft 8.1 and 8.4.

Business components, component interfaces, and records

The connector requires a PeopleSoft Business Component and Component Interface for each hierarchical business object that it processes.

The adapter does not include PeopleSoft-specific business objects or the Component and Component Interface that must be associated with each business object. These objects must be created by the person who implements the connector. However, to assist in business-object development, the adapter includes a sample PeopleSoft-specific business object. This sample is located in the `connectors\PeopleSoft\samples` directory within the product directory. For information on creating a Component Interface and its corresponding business object, see "Component interface and business object relationship" on page 30. For information on creating the required classes and methods, see "Generating APIs" on page 35.

Note: In this document backslashes (\) are used as the convention for directory paths. For UNIX installations, substitute slashes (/) for backslashes (\). All file pathnames are relative to the directory where the product is installed on your system.

Event-processing components

To enable event notification, the adapter provides the CW_EVENT_vX Project, which includes:

- Fields

  The fields store event information.

- Records

  The records include an event table (CW_EVENT_TBL), an archive table (CW_ARCHIVE_TBL), and a function library (FUNCLIB_CW). The function library contains the connector-specific event-notification functions that are called from Components and Records during `SavePostChg()` PeopleCode events. For more information, see "Processing application events" on page 6, "cw_publish_events() Function" on page 19, and "Event and archive tables" on page 20.

- Component

  The CW_EVENT_BC Component enables events to be viewed online. It also provides the appropriate structure for the Component Interface required for event processing. The Component Interface exposes the fields from the CW_EVENT_TBL record that are required for the connector's processing. The CW_EVENT_BC Component contains the CW_EVENT_TBL record.

- Component Interface

  The CW_EVENT_CI Component Interface exposes the fields and records of the component as the properties and methods necessary for event processing.

- Menu Definition

  The CW_EVENT_MNU Menu Definition enables event pages to be displayed online.

How the connector works

This section describes:

- "Interacting with the PeopleSoft application"
- "Processing business object requests" on page 4
- "Processing application events" on page 6

Interacting with the PeopleSoft application

At startup, the connector creates a session object through which it connects to the PeopleSoft Application Server. Connecting to the Application Server gives the connector access to the APIs for all the Component Interfaces that correspond to its supported business objects. The server also provides access to the PeopleCode and the Application Designer objects included with the adapter for event notification.

Each Component Interface (and its associated Business Component, Records, Fields, Scrolls, and PeopleCode) contains all the information required by the connector to process a hierarchical WebSphere business object for PeopleSoft. Because each Component Interface encapsulates its Business Component's data and processing logic, the connector does not replicate this processing logic. For example, the connector need not explicitly handle duplicate record checks, edit-table validations, or security.

If an error occurs within the connector or during its online processing within the PeopleSoft application, the connector's application-specific component sends the `FAIL` return code to the connector framework, which sends it to the integration broker. If the connector loses its connection to the Application Server, the

connector's application-specific component sends the return code of APPRESPONSETIMEOUT, and then the terminate() method is invoked on it.

For information on how the connector processes data in a business object, see Chapter 3, "Understanding business objects for the connector," on page 29

Processing business object requests

When the connector receives a business object request to change data in the application, the connector processes hierarchical business objects recursively. In other words, the connector processes each child business object until it has processed data for all levels in the Component Interface associated with the business object.

When processing a business object request from the integration broker, the connector calls PeopleSoft APIs from the session object in the following order:

1. The connector uses the getComponent("ciName") method to return the Component Interface associated with the business object. The Component Interface's name is stored in the application-specific information property at the business-object level of each business object.
2. If the business object requests data creation or modification, the connector inserts or changes values in the application. If the business object requests data retrieval, the connector obtains values from the application.
   - For a Create or Update request, the connector uses the set*FieldName*(*value*) method to set each field on the Component Interface for which it obtained a value in step 1.

     If the application-specific information of a key attribute specifies that the PeopleSoft application generate the unique ID, the connector sends the business object to the application with the string NEXT specified as the value of the attribute. For information on using the NEXT string in a WebSphere business object for PeopleSoft, see "Application-specific information at the attribute level" on page 49.

     If the business object's application-specific information defines setInteractiveMode as true, entering or exiting each field triggers the associated business logic in the underlying component, which immediately publishes errors to the PeopleSoft PSMessage collection queue.
   - For a Retrieve request, the connector uses the get*FieldName*() method without specifying a value.

   For information on how the connector processes data in a business object, see Chapter 3, "Understanding business objects for the connector," on page 29. For information on connector-specific properties, see "Connector-specific properties" on page 22.
3. After processing a Create or Update request for an entire business object, including all its child business objects, the connector invokes the Component Interface's Save() method. If successful, the Save() method executes a single COMMIT statement. If the business object's application-specific information defines setInteractiveMode as false, invoking the Save() method triggers all FieldEdit business logic associated with the records and fields of the underlying component. All PeopleCode errors are published to the PSMessage collection queue.
4. If the PeopleSoft system generates the unique identifiers (IDs) during a Create request, the connector uses standard PeopleSoft auto-numbering functionality (that is, built-in functions) to retrieve the most recently used ID, and populate the business object with a new one.

5. The connector's processing uses all the business logic that is provided in the associated Component Interface.

If a connection error is detected when the connector is processing a business object request, the connector's application-specific component logs a fatal error and sends the return code of `APPRESPONSETIMEOUT` to trigger email notification. The connector is then terminated.

The following sections describe request verb processing:

- "Processing create requests"
- "Processing retrieve requests" on page 5
- "Processing update requests" on page 5
- "Processing delete requests" on page 6
- "Processing existence requests" on page 6

Processing create requests

When the integration broker sends a business object request with the Create verb, the connector uses PeopleSoft's `Create()` method to create a new instance of the Component Interface. If the instance passes all PeopleSoft business logic as the transaction proceeds, it is saved in the application. The object created in the application contains all values contained in the business object, including all child business objects.

For more information about processing a create operation, see "Create operations" on page 40.

Processing retrieve requests

When the integration broker sends a business object request with the Retrieve verb, the connector uses PeopleSoft's `Get()` method to verify that a unique instance of the corresponding Component Interface exists. This `Get()` method instantiates the Component Interface, allowing the connector to load its values into the business object. The business object that the connector returns to the integration broker exactly matches the instance of the Component Interface.

In other words, the value of each simple attribute of the business object returned to the integration broker matches the value of the corresponding Property Field in the Component Interface. Also, if the returned business object is hierarchical, the number of individual business objects in each of its arrays matches the number of levels or collections in the Component Interface for that array.

For more information about processing a retrieve operation, see "Retrieve operations" on page 41.

Processing update requests

When the integration broker sends a business object request with the Update verb, the connector modifies an existing instance of the Component Interface. If the instance passes all PeopleSoft business logic as the transaction proceeds, it is saved in the application.

The object updated in the application exactly matches the request business object. The connector updates all simple Property Fields except those whose corresponding attribute in the request business object contain the value `CxIgnore`. It inserts all child business objects contained in the request business object.

Depending on the value of its KeepRelationship application-specific information parameter, the connector either deletes or retains child business objects that do not exist in the request business object.

For more information about processing an update operation, see "Update operations" on page 43.

Processing delete requests

Because PeopleSoft does not support deletion of transactions, the connector does not either. The standard behavior for processing a logical delete is to use the Update verb to change the status of the business object's EffectiveStatus attribute to "I" (Inactive).

However, to cause the connector to support a delete of an entire object, do the following:

1. Create a user-defined PeopleCode method that performs deletion within PeopleSoft.
2. Expose the method through the appropriate Component Interface.

Processing existence requests

When the integration broker sends a hierarchical business object request with the Exists verb, the connector checks for an instance of the Component Interface. The connector's application-specific component returns SUCCEED if the instance exists in the application, and FAIL if the object does not exist.

Processing application events

Event notification involves three main processes:

- "Event publication"—Events are published to the connector's event table (CW_EVENT_TBL), which is stored in the PeopleSoft database.
- "Event polling"—The connector polls the event table through the PeopleSoft API.
- "Event archiving" on page 7—The connector archives events into its archive table (CW_ARCHIVE_TBL).

Event publication

To publish events to the connector's event table, the PeopleSoft application uses PeopleCode and Application Designer objects included with the adapter. For information about these objects, see "Event-processing components" on page 3.

The event publication process uses the cw_publish_events() function, which is stored in the FieldFormula event of the FUNCLIB_CW record. You must declare and call this function from the SavePostChg() PeopleCode of the component involved in the event.

Note: This procedure does not use a Component Interface.

For more information, see "cw_publish_events() Function" on page 19.

Event polling

The connector polls the event table at a regular, configurable interval. It searches for events first by status, then by the value of the connector property ConnectorID. If this value is null or blank, the property will not be used for the search. When the connector processes an event, it immediately updates the status to INPROGRESS (a value of 3). Any pending INRPOGRESS events are reset to READYFORPOLL (a value of 0) during the connector's initialization.

The connector uses the CW_EVENT_CI Component Interface's `Find()` method to poll the event table. This method returns a Collection of events with the status `READYFORPOLL`. The connector loops through the Collection, obtaining the name of every business object listed in the CW_EVENT_TBL.

The connector uses the API to set and get property values, gathering event information for each event returned in the Collection. The connector checks to determine which business objects are subscribed. For subscription information specific to your integration broker, see the broker's implementation guide.

- For each subscribed business object, the connector retrieves the name of its associated Component Interface. Using the key values listed in the CW_OBJ_KEYS field, the connector instantiates the Component Interface involved in the event and copies the information into an application-specific business object. After creating the business object, the connector passes it to the integration broker and archives the event.
- For unsubscribed business objects and business objects that generate errors, the connector moves the event to the archive table and updates its status.
- The connector supports the delete verb for outbound objects. The key values of the business object instance are set, and the object is sent to the WBI Integration broker.

For more information, see "Event and archive tables" on page 20.

Event archiving

While the CW_EVENT_CI Component Interface is instantiated, the connector calls the `cw_archive_events()` user-defined method for each event it processes. This method, which is in the form of PeopleCode, sets the Archive flag to `Y`. Setting this flag causes the `SavePostChg()` PeopleCode to archive the event when the user invokes the `Save()` method. The archive table also records the date and time the event was processed.

Note: The `SavePostChg()` PeopleCode is included with the Project.

You can use the archive table to query for event history or to troubleshoot problems in event processing. For example, all unsubscribed events have a status of `unsubscribed`.

Because there are potential failure points associated with the processing of events, the event management process does not delete an event from the event table until it has been inserted into the archive table.

Event notification process flow

Figure 2 illustrates the event notification process flow.

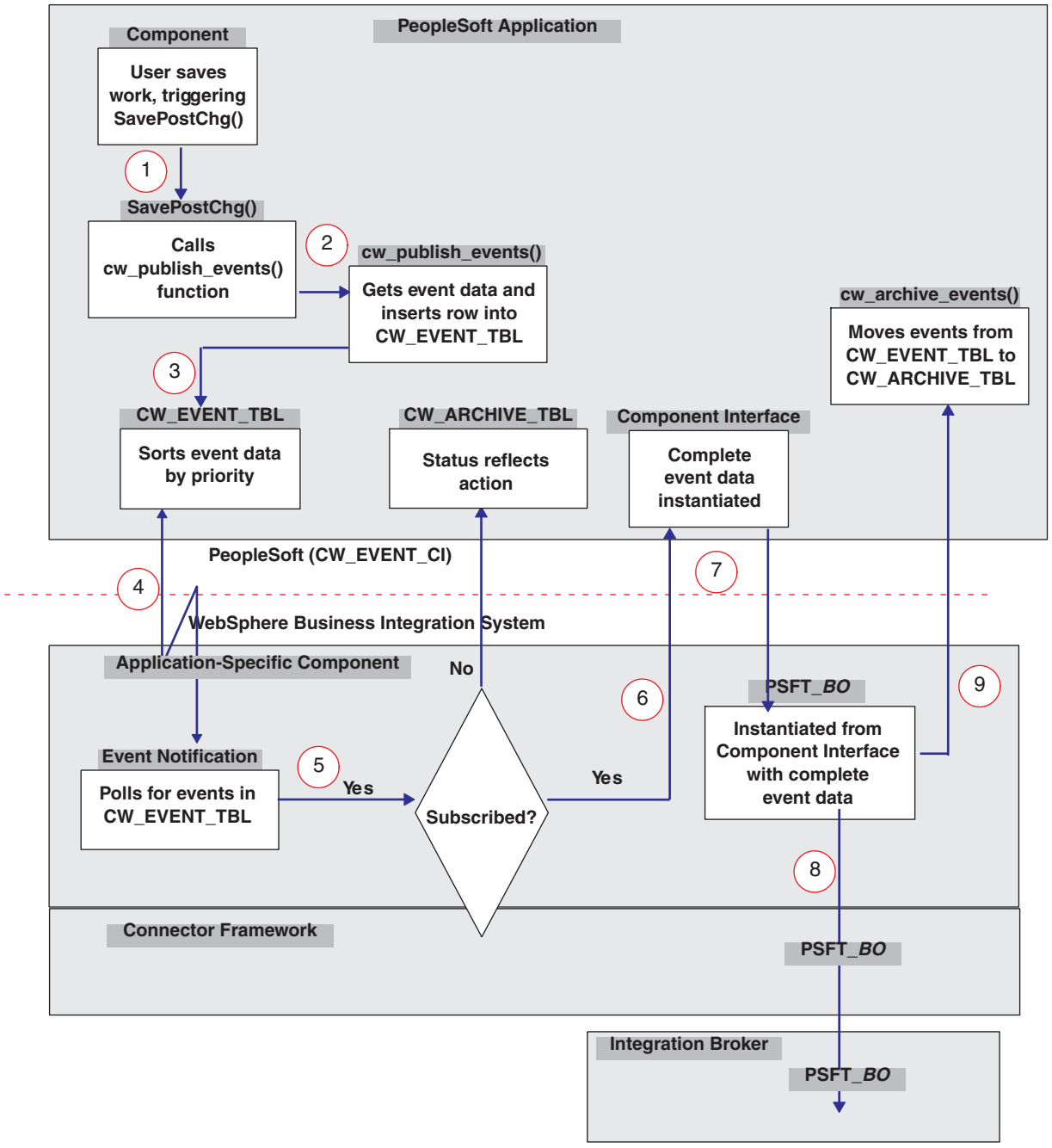

Figure 2. Event notification process flow

The following describes the steps illustrated in Figure 2:

1. When a user saves a change online in a Component, the `SavePostChg()` method calls the `cw_publish_events()` function.
2. The `cw_publish_events()` function, which has four parameters, does the following:
   - Gets the application-specific business object name from its first parameter.
   - Gets the name of each key required to instantiate the Component Interface that corresponds to the application-specific business object. It gets the key names and their respective values from its second parameter.
   - Sets the priority of the event based on the value of the third parameter.

Note: Because the application can publish multiple events from a single Save action on a Component, the connector uses the value of the third parameter to prioritize events. If sequence order is important to an object, it should specify a priority when it calls the `cw_publish_events()` function. For example, a Location would have a higher priority than a Department because the Location must be processed first.

- Sets the `CW_CONN_ID` field to the value of the fourth parameter, which also corresponds to the `ConnectorID` connector property. The default is `PeopleSoftConnector`.
- Generates the unique ID for the event.
- Evaluates the PeopleSoft system variable `%Mode` to identify the verb associated with the event.
- Inserts the event into CW_EVENT_TBL.

For more information, see "cw_publish_events() Function" on page 19.

3. The connector sorts events in CW_EVENT_TBL by priority.
4. The connector polls the event table for events at the interval specified by its PollFrequency configuration property, picking up no more than the number of events specified by its PollQuantity configuration property. To do the polling, the connector instantiates the CW_EVENT_CI Component and invokes the `Find()` method.

   Note: The connector uses a different thread from the one used for request processing.
5. For each record retrieved from the event table, the connector checks whether it subscribes to the event data.

   If the data is not subscribed, the connector moves the event record from CW_EVENT_TBL to CW_ARCHIVE_TBL with a status of `unsubscribed`.
6. If the data is subscribed, the connector uses the event record's key fields to instantiate the appropriate Component Interface, from which it retrieves complete data for the event.
7. After retrieving complete event data from the Component Interface, the connector instantiates the appropriate application-specific business object.
8. The connector sends the application-specific business object to the integration broker.
9. The connector calls the `cw_archive_events()` function, which sets the Archive flag to `Y` and sets the status and processing time appropriately. Setting the Archive flag causes the `SavePostChg()` PeopleCode to copy the event information to the archive table and delete the event from the event table.

Processing locale-dependent data

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

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

components in the WebSphere business integration system are written in Java. Therefore, when data is transferred between most integration components, there is no need for character conversion.

To log error and informational messages in the appropriate language and for the appropriate country or territory, configure the `Locale` standard configuration property for your environment. For more information on configuration properties, See Appendix A, “Standard configuration properties for connectors,” on page 79.

Chapter 2. Installing and configuring the connector

This chapter describes how to install and configure the connector component of IBM WebSphere Business Integration Adapter for PeopleSoft and how to configure the PeopleSoft application to work with the connector.

This chapter contains the following sections:

Compatibility

The adapter framework that an adapter uses must be compatible with the version of the integration broker (or brokers) with which the adapter is communicating. The 2.3.x version of the Adapter for PeopleSoft Guide is supported on the following adapter framework and integration brokers:

- **`Adapter framework`**: WebSphere Business Integration Adapter Framework, versions 2.1, 2.2, 2.3.x, and 2.4.
- **`Integration brokers:`**
  - WebSphere InterChange Server, version 4.11, 4.2, 4.21, and 4.22.
  - WebShpere MQ Integrator, version 2.1.0
  - WebSphere MQ Integrator broker, version2.1.0
  - WebSphere Business Integration Message Broker, version 5.0
  - WebSphere Application Server Enterprise, version 5.0.2, with WebSphere Studio Application Developer Integration Edition, version 5.0.1

See the Release Notes for any exceptions.

Note:

For instructions on installing the integration broker and its prerequisites, see the following documentation.

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

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

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

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

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

Prerequisites

Before you use the IBM WebSphere Business Integration Adapter for PeopleSoft, verify that your system has the required software. Also, you must perform operations such as creating the connector's user account, mapping an environment variable, and generating PeopleSoft APIs. This section describes:

- "Required software for using the connector"
- "Required operations for installing the connector" on page 13

Required software for using the connector

To access a designated Component Interface and its underlying PeopleSoft business logic, the connector requires two layers of PeopleSoft API classes. Before you use the connector, install the following:

- `psjoa.jar`— The connector uses this API to connect to the Application Server through the BEA Systems' Jolt port. The connector also uses this API to send objects through the Application Server. This file is included with the current version of PeopleTools and is located in the following directory:

  UNIX

  ```
  $PS_HOME/web/PSJOA
  ```

  End of UNIX

  Windows

  ```
  %PS_HOME%\web\PSJOA
  ```

  End of Windows

  Save this file to the directory documented in "Installed file structure on Windows" on page 15 and "Installed file structure on UNIX" on page 14. For more information on retrieving PeopleSoft's API files, see PeopleSoft's *Installation and Administration Guide*.

- Component Interface API—The connector accesses this layer only after a session object has been created and a connection to the PeopleSoft Application Server has been established using `psjoa.jar`. The Component Interface API provides access to all of a Component Interface's exposed objects and PeopleCode methods. You must manually generate this API in Application Designer, as described in "Generating APIs" on page 35.

Important: For installation prerequisites specific to your integration broker, see the For more information on installing the connector component, refer to the *Implementation Guide for the WebSphere MQ Integrator Broker,* or, the *WebSphere InterChange Server System Installation Guide for Windows,* or *for Unix*.

Required operations for installing the connector

This section describes required operations that must be performed before you use the connector:

- Map the CLASSPATH environment variable to the following directories:

UNIX

```
$ProductDir/connectors/PeopleSoft/dependencies
```

End of UNIX

Windows

```
%ProductDir%\connectors\PeopleSoft\dependencies
```

End of Windows

  Also ensure that psjoa.jar is defined in the CLASSPATH.
- Create a user account in PeopleSoft for the connector. For more information, see "Creating a user account in PeopleSoft."
- Configure the PeopleSoft user account so that it does not time out. For more information, see "Configuring time-out for the user account" on page 13.
- Only when ICS is the integration broker: Delete any previously installed versions of the connector. Also, delete all previous customizations made in PeopleSoft for the connector. For example, delete all connector-related Menus, Pages, Records, Message Definitions Components, and Component Interfaces. Also, delete the connector's event and archive tables. Typically, these objects are stored in a connector-specific project that you can open in Application Designer.

  Note: If you install a WebSphere Business Integration Adapter on the same machine as a previous installation of the connector for ICS, the ICS connector will not run unless you manually switch the IBM WebSphere system environment variable to point to the desired product path.
- Generate Component Interface APIs. For more information, see "Generating APIs" on page 35.

Creating a user account in PeopleSoft

Before installing the connector, you must create a user account in the PeopleSoft application. The account:

- Can have any valid PeopleSoft user name and password.
- Must have privileges to retrieve, insert, update, and delete data from the appropriate Pages, Components, and Component Interfaces in the PeopleSoft system. For example, for the connector to process customer data, the connector user account must have privileges to work with relevant Customer components.
- Uses General PeopleSoft Security. For more information, see the Security section of Administrator Tools in PeopleBooks.
- Must be configured so that the session never times out. For more information, see "Configuring time-out for the user account."

Configuring time-out for the user account

Because the connector uses the Component Interfaces (and the Component Processor on the Application Server) to insert and extract business objects from the PeopleSoft database server, the Application Server must be running. If the user account times out, the Component Processor closes, and the connector logs an

error message every time it attempts to access the PeopleSoft system (that is, at every poll, at every create, at every retrieve, and at every update).

To configure the user account to never time out, do the following:

1. In the PeopleSoft application, select `PeopleTools` from the `Go` menu option.
2. Select `Maintain Security` from the `PeopleTools` menu option.
3. Select `Permission Lists` from the `Use` menu option.
4. Select `Sign-On Times` from the `Permission Lists` menu option
5. From the list of permissions, select the one affiliated with the connector's user account.
6. On the `General` tab, select `Never Time-Out`, and select `Can Start Application Server`, if not already selected.

Installing the adapter and related files

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

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

Installed file structures

Installed file structure on UNIX

Table 1 describes the UNIX file structure used by the connector.

Table 1. Installed UNIX file structure for the connector

Subdirectory of $*ProductDir***S**	**Description**
`connectors/PeopleSoft`	• Contains the connector's `CWPeopleSoft.jar` and the `start_PeopleSoft.sh` files. The startup script for the connector is called from the generic connector manager script. When you click Install from the product installer's Connector Configuration screen, the installer creates a customized wrapper for this connector manager script. When the connector works with WebSphere Interchange Server broker, use this customized wrapper to start and stop the connector. When the connector works with WebSphere MQ Integrator broker, use this customized wrapper only to start the connector; use `mqsiremotestopadapter` to stop the connector. • Contains an executable named `EventNotificationInstall.exe`, which contains the event notification objects • Contains a file named `SavePostChg.txt`, which contains a sample of the PeopleCode required to process events
`/bin`	Please verify that this folder contains the `CWConnEnv.sh` file.
`/lib`	Please verify that the directory contains the `WBIA.jar` file. This file corresponds to ADK 2.2.0. If an older version of ADK is required, please refer to the *Connector Development Guide.*

Table 1. Installed UNIX file structure for the connector (continued)

Subdirectory of $*ProductDir***S**	**Description**
`connectors/PeopleSoft/dependencies`	• Contains the `psjoa.jar` files that you must copy from the current version of PeopleTools. • Contains the `PSFTCI.jar` file, which contains all the PeopleSoft Component Interface class files that PeopleSoft generates after you build and compile the APIs.
`connectors/messages`	Contains the `PeopleSoftConnector.txt` file, as well as `PeopleSoftConnector_ll_TT.txt` files (message files specific to a language (*_ll*) and a country or territory (*_TT*)).
`connectors/PeopleSoft/samples`	Contains a sample PeopleSoft-specific business object.
`repository/PeopleSoft`	Contains the `CN_PeopleSoft.txt` file.

Installed file structure on Windows

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

Table 2. Installed Windows file structure for the connector

Subdirectory of%*ProductDir***S%**	**Description**
`\connectors\PeopleSoft`	• Contains the connector's `CWPeopleSoft.jar` and the `start_PeopleSoft.bat` files. • Contains an executable named `EventNotificationInstall.exe`, which contains the event notification objects • Contains a file named `SavePostChg.txt`, which contains a sample of the PeopleCode required to process events
`..\bin`	Please verify that this folder contains the `CWConnEnv.bat` file.
`..\lib`	Please verify that the directory contains the `WBIA.jar` file. This file corresponds to ADK 2.2.0. If an older version of ADK is required, please refer to the *Connector Development Guide.*
`\connectors\PeopleSoft\dependencies`	• Contains the `psjoa.jar` files that you must copy from the current version of PeopleTools. • Contains the `PSFTCI.jar` file, which contains all the PeopleSoft Component Interface class files that PeopleSoft generates after you build and compile the APIs.
`\connectors\messages`	Contains the `PeopleSoftConnector.txt` file, as well as `PeopleSoftConnector_ll_TT.txt` files (message files specific to a language (*_ll*) and a country or territory (*_TT*)).
`\connectors\PeopleSoft\samples`	Contains a sample PeopleSoft-specific business object.
`\repository\PeopleSoft\`	Contains the `CN_PeopleSoft.txt` file.

Enabling the application for the connector

Before you can use the connector with your PeopleSoft application, you must:

- Define required Components, Component Interfaces, Menus, Pages, Records, and PeopleCode in your PeopleSoft application. To facilitate the process of defining required business objects, the adapter includes a sample business object. For

more information, see Chapter 3, "Understanding business objects for the connector," on page 29 and Chapter 4, "Generating business object definitions using PeopleSoftODA," on page 53.

- Import the Project included with the adapter. This Project contains the components required for event processing, build the required tables, and build the API files.

  Note: Importing the Project is necessary only when you use the connector to process events.

This section describes:

- "Installation operations required to process application events" on page 16
- "Code for processing application events" on page 19

Installation operations required to process application events

This section describes required operations that must be performed when you use the connector to process application events. Perform this installation before you use the connector for the first time.

Installing event-processing components involves the following processes:

- "Importing the project" on page 16
- "Building required objects" on page 17
- "Building the API files" on page 18

Importing the project

1. Run the file that unzips the Application Designer project files into the designated folder. From the *%ProductDir*S%\connectors\PeopleSoft\dependencies directory, select the EventNotificationInstall.exe file.

   Note: This file is available only on Windows.

   Running this file with the default values creates a Projects directory within the dependencies directory. The project name is CW_EVENT_*Vx*, where *Vx* identifies the version number.
2. Open PeopleSoft's Application Designer and select: Copy Project From File... from the File menu.
3. In the menu that appears, select the import directory into which the project was unzipped. Figure 3 illustrates the screen within PeopleSoft.

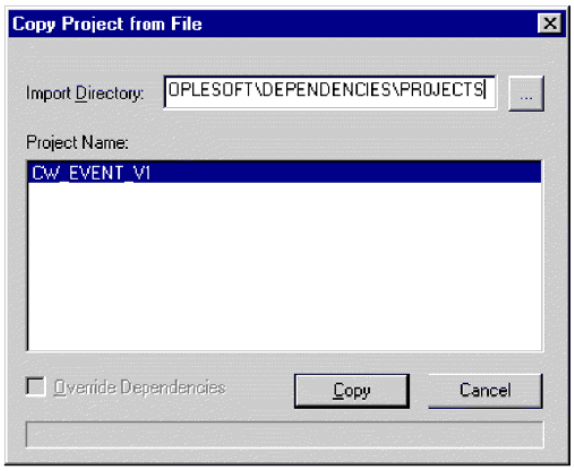

Figure 3. Importing the project

4. Click Copy and verify the list of components with those listed in "Event-processing components" on page 3.

Building required objects

To build the event and archive tables and the function library:

1. Build the tables by selecting `Project...` from the `Build` menu.
2. In the dialog box that appears, verify that the CW_EVENT_TBL, FUNCLIB_CW and CW_ARCHIVE_TBL appear as illustrated in Figure 4.

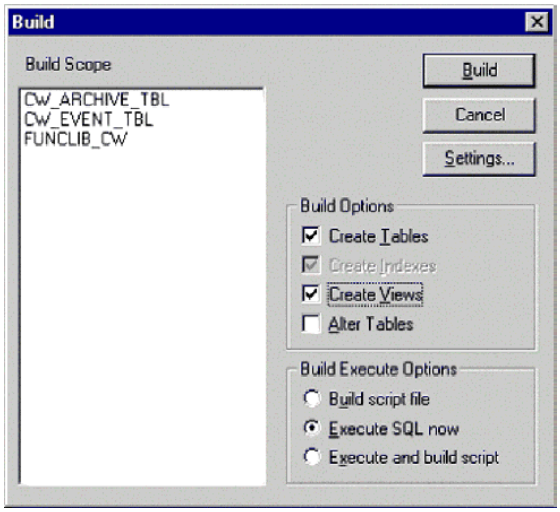

Figure 4. Building the tables

3. Select the build options "Create Tables" and "Execute SQL now". Also, click the Settings button and, in the window that displays, verify that "create table if it already exists" and "recreate index if it already exists" are selected.
4. Click Build.
5. Log on to a SQL editor using an ID with appropriate database privileges. The default ID is `SYSADM/SYSADM`.Initialize the CW_EVENT_NOT Field on the FUNCLIB_CW Record with a value of zero. To do so, run the following statement:

```
INSERT INTO PS_FUNCLIB_CW (CW_EVENT_NOT) VALUES ('0');
```

Building the API files

To build the CW_EVENT_CI API files (required when the connector processes only events) or the Component Interface API files (required when the connector processes both events and requests):

1. Open the Component Interface CW_EVENT_CI from the project window and select `PeopleSoft APIs` from the `Build` menu.
2. In the Java Classes panel:
   - Select the Build box
   - In the "Directory containing PeopleSoft package:" field, enter the path for connector's `dependencies` directory (or the directory where all the PeopleSoft API files reside for your implementation). This is usually:

UNIX

```
$ProductDirS/connectors/PeopleSoft/dependencies
```

End of UNIX

Windows

```
%ProductDirS%\connectors\PeopleSoft\dependencies
```

End of Windows

3. From the "Select APIs to Build:" field, select CWEVENT_CI and its associated collection. Figure 5 illustrates the screen within PeopleSoft.

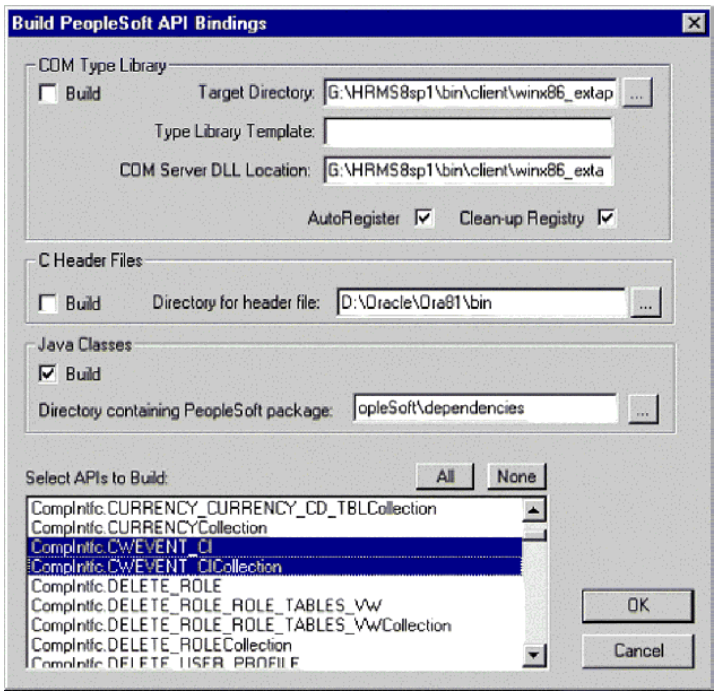

Figure 5. Building APIs

4. Click OK.
5. Compile any API files that you generate, and add them to `PSFTCI.jar` (if it exists) or create this file if it does not exist.

Note: The start script or batch file is configured to search for these API files in a file named PSFTCI.jar in the \connectors\PeopleSoft\dependencies directory. After you compile these API classes, make a jar file that you name PSFTCI.jar. If you put this jar file in a different directory, change the start script or batch file to point to the correct location of these API classes.

6. Place the cw_publish_events() function declaration and function call in the SavePostChg() PeopleCode for each Component associated with an event. For more information, see "Sample PeopleCode declaration and function call" on page 20.

   Note: The function must be placed in the SavePostChg() of the Component and not of the Record.

7. Define all parameters used in the function call and insert PeopleCode to check whether the Component has changed before making the function call. This code eliminates unnecessary calls to the function. Also add a check for %userid to prevent the connector from ping-ponging (that is, creating an endless loop in which the connector interprets a data change from a request as a new application event). For an example, see "Sample PeopleCode declaration and function call" on page 20.

Code for processing application events

The adapter includes the cw_publish_events() function in the FUNCLIB_CW function library. This function inserts events into the connector's event table. This section describes the function and the sample code that calls it.

cw_publish_events() Function

The cw_publish_events() function takes four parameters, all of type String:

- &BONAME—contains the name of the WebSphere application-specific business object for PeopleSoft that is generated for the event
- &KEYLIST—contains the name of all keys required to instantiate the Component Interface that corresponds to the business object named in the &BONAME parameter. This function uses the GetKey() method for instantiation. Because the connector uses name-value pairs, the order of the keys is not significant. However, the names must match those listed on the Component Interface. Separate multiple keys with a colon or other configurable delimiter, for example, SETID:DEPTID. For more information, see "EventKeyDelimiter" on page 23.
- &CWPRIORITY1—defines the priority of the event; defaults to 2. This parameter enables certain events to be processed prior to events with a lower specified priority. The CW_EVENT_TBL processes events of higher priority first.
- &CONNID—contains the name of the connector instance that will retrieve the event. This parameter is used for event distribution and defined in the ConnectorID property for the connector. If not using event distribution, use the value "PeopleSoftConnector".

Important: You must define these parameters, in proper form, before or during the function call.

Using the values specified for its parameters and the information currently available in the Component Buffer, the function gathers the required information from the Component and inserts the event in the event table. The function performs the following:

- Evaluates the PeopleSoft system variable %Mode, which contains values such as Add and Update/Display, to identify the verb associated with the event

- Uses the key names to gather the current key values from the Component and creates a list of business object key names and values
- Loads all information into a row, which it inserts into the event table using PeopleCode built-in functions and methods

Sample PeopleCode declaration and function call

The adapter includes a sample of the PeopleCode declaration and function call in the `savepostchg.txt` file in the connector's `samples` directory. Replace the business object's name and keys with the correct information for your business object and then copy and paste the code directly into the PeopleCode editor.

Before making the actual function call, use a simple logic test to verify that the Record or Component actually changed. If it did not change, the connector does not call the function, which enhances performance. Also verify that the `%userid` is not `CW`. Doing so prevents the connector from interpreting a data change from a request as a new application event.

The following code is the sample that the adapter provides:

```
/* Place this code in Component's SavePostChg() and define the four */
/* parameters used in the function call */

Declare Function cw_publish_event PeopleCode FUNCLIB_CW.CW_EVENT_NOT
 FieldFormula;
Component String &BONAME1;
Component STring &KEYLIST1;
Component String &CWPRIORITY1;
Component String &CONNID1;

   &BONAME1 ="Psft_Dept";
   &KEYLIST1 = "DEPT_TBL.SetId:DEPT_TBL.DeptId";
   &CWPRIORITY = 2;
   &CONNID1 ="PeopleSoftConnector";

/* Check if Component Changed before calling function */
If ComponentChanged() and
    %userid <> "CW" then

   /* Publish this event to the IBM WebSphere
   CW_EVENT_TBL for polling */

   cw_publish_event(&BONAME1,&KEYLIST1,&CWPRIORITY1,&CONNID1);

End-if;
```

Event and archive tables

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

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

- Event was processed successfully
- Event was not processed successfully
- Event was not subscribed to

If events are not deleted from the event table after the connector picks them up, they occupy unnecessary space there. However, if they are deleted, all events that are not processed are lost and event-processing cannot be audited. Therefore, the adapter provides the archive table to store events deleted from the event table.

Table 3 describes the columns in the event and archive tables.

Table 3. Event and archive table schema

Name	Description	Type	Constraint
CW_EVENT_ID	Internal identifier of the event A unique key field that identifies each event generated from within PeopleSoft	NUMBER	Primary key
CW_CONNECTOR_ID	Unique ID of the connector for which the event is destined. This value is important when multiple connectors poll the same table.	VARCHAR	
CW_OBJ_KEYS	Keys of the business object, specified in name-value format. When defining parameters for a function call (such as `cw_publish_events`), the name consists of the table name and field name separated by a period. Multiple keys are separated with a colon or other configurable delimiter, for example: `DEPT_TBL.SetId:DEPT_TBL.DeptId`For more information, see "EventKeyDelimiter" on page 23.	VARCHAR	Not null
CW_OBJ	Name of the business object	VARCHAR	Not null
CW_VERB	Verb associated with the event. A PeopleCode function included with the adapter determines the value (Create, Retrieve, Update, Delete) based upon the `%Mode` system variable used by PeopleSoft	VARCHAR	Not null
CW_PRIORITY	Event priority (0 is highest, n is lowest), which the connector uses to get events on a priority basis.	NUMBER	Not null
CW_DTTM	Date and time the event or archiving occurred	STRING	Default current date/time (for archive table, actual event time). PeopleSoft treats the datatype for DATE fields as STRING and returns the same.
CW_STATUS	-2 (Error sending event to integration broker) -1 (Error processing event) 0 (Ready for poll) 1 (Sent to integration broker) 2 (No Subscriptions for the business object) 3 (In Progress). This status is used only in the event table and not in the archive table.	VARCHAR	Not null

Configuring the connector

You must configure the connector's standard and connector-specific connector configuration properties before you can run it. To configure connector properties, use:

- Connector Designer—if WebSphere InterChange Server is the integration broker
  Access this tool from the System Manager.

- Connector Configurator—if WebSphere MQ Integrator broker is the integration broker

  Access this tool by clicking its menu option in the submenu of the Windows Start menu.

As you enter configuration values, they are saved in the repository.

Standard connector properties

Standard configuration properties provide information that all connectors use. See Appendix A, "Standard configuration properties for connectors," on page 79 for documentation of these properties.

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

Table 4 provides information specific to this connector about a configuration property in the appendix.

Table 4. Property information specific to this connector

Property	Note
`AgentConnections`	Because this connector is single-threaded, do not change the default value of this property.
`CharacterEncoding`	This connector does not use this property.
`Locale`	Because this connector has been internationalized, you can change the value of this property. See release notes for the adapter to determine currently supported locales.

Connector-specific properties

This section documents configuration properties specific to this connector at runtime. Connector-specific properties provide a way of changing static information or logic within the connector without having to recode and rebuild it.

Table 5 lists the connector-specific configuration properties for the connector. See the sections that follow for explanations of the properties.

Table 5. Connector-specific configuration properties

Name	Possible values	Default value	Required
AppServerMachineNameOrIP		n/a	Yes
ApplicationPassword	*Password for the connector's user account*	PS	Yes
ApplicationUserName	*Name of the connector's user account*	PS	Yes
ConnectorID	*Name of connector for event distribution*	null	No
ConvertToPrimitiveFloat	true or false	false	No
EventKeyDelimiter	<NameValueChar><*DelimiterChar*> Note: Do not specify a space between the actual characters.	=:	No
PingCompInterface	replace < CHANGEME > with the name of an existing component interface	CHANGME	Yes
PollQuantity	Values are 1 to 500	1	No
PortNumber	*The Jolt port number*	9000	Yes
ReconnectSessionOnGetFail	true or false	false	No
	Values are 1or greater	1	No
SetLangCode	true or false	false	No
UseDefaults	true or false	false	Yes
Version	PeopleTools version number	8.15	Yes

AppServerMachineNameOrIP
The name or IP address of the machine where the Application Server is running. The connector uses this value only when connecting to the application.

ApplicationPassword
Password for the connector's user account in the PeopleSoft application.

There is no default value.

ApplicationUserName
Name of the connector's user account in the PeopleSoft application.

There is no default value.

ConnectorID
The value of this property is used to search for events in the CW_EVENT_TBL through the cweventci. It is used for event distribution, when multiple connectors are used to retrieve specific events. If the value is left blank or null, this property will not be set prior to invoking the Find() method in the event component interface.

ConvertToPrimitiveFloat
Specifies whether to convert float objects to primitive float objects.

The default value is false.

EventKeyDelimiter
Specifies the two characters in name-value pairs that separate the name from its value (NameValue character) and the pairs from each other (*Delimiter* character). The following example uses the default equals (=) *NameValue* character and default colon (:) *Delimiter* character:

```
SETID=1234:DEPTID=5678
```

The default value is =:

PIngCompInterface
Pings the component interface to test for network connectivity/session validity. FAIL is returned when either the session or connectivity are invalid. When the name of the component interface is given, the connector agent terminates if the session instance is unable to get that interface, because of either invalid session or network connectivity issues.

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

The default is 1.

PortNumber
The Jolt Port Number (not the Tuxedo port number) on the Application Server. The connector connects to the `JSL`, not to the `WSL`.

The default value is `9000`

ReconnectSessionOnGetFail
If set to `true`, the connector automatically creates a new session object and reconnects to the PeopleSoft application. The connector uses this property only when the component interface's `Get()` method returns a non-critical error or warning message that terminates the connector's session object. Typically this problem arises when an instance has different keys from those used by the connector.

The default value is `false`.

SessionPoolSize
Enables the connector to run multiple PeopleSoft session instances. Set this property to the number of session instances you want to run. Multi-threading is only available to inbound objects. Each inbound object thread is allocated a session instance from a pool of free sessions, and then that session instance moves to a busy pool. When the transaction finishes, the session instance is released back to the free pool.

When the connector runs multi-threaded, one session instance is reserved for polling which is single-threaded and performed serially.

SetLangCode
Specifies whether the connector sets the base language immediately after it connects to the PeopleSoft application. If set to `true`, the connector uses a workaround that correctly updates the base language in the base table rather than in the language table. If you set this property to `false`, your base tables may not be updated correctly.

The default value is `false`.

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

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

The default value is false.

Version

Specifies the current version of PeopleTools on which the application is running. The value should be specified to the latest available decimal point.

The default value is 8.16.

Creating multiple connector instances

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

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

Create a new directory

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

```
ProductDir\connectors\connectorInstance
```

where `connectorInstance` uniquely identifies the connector instance.

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

```
ProductDir\repository\connectorInstance
```

Create business object definitions

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

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

   ```
   ProductDir\repository\initialConnectorInstance
   ```

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

Create a connector definition

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

1. Copy the initial connector's configuration file (connector definition) and rename it.

2. Make sure each connector instance correctly lists its supported business objects (and any associated meta-objects).
3. Customize any connector properties as appropriate.

Create a start-up script

To create a startup script:

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

   dirname
2. Put this startup script in the connector directory you created in "Create a new directory" on page 25.
3. Create a startup script shortcut (Windows only).
4. Copy the initial connector's shortcut text and change the name of the initial connector (in the command line) to match the name of the new connector instance.

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

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

Starting the connector

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

ProductDir\connectors*connName*

where *connName* identifies the connector. The name of the startup script depends on the operating-system platform, as Table 6 shows.

Table 6. Startup scripts for a connector

Operating system	Startup script
UNIX-based systems	connector_manager_*connName*
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 Adapters>Adapters>Connectors**. By default, the program name is "IBM WebSphere Business Integration Adapters". However, it can be customized. Alternatively, you can create a desktop shortcut to your connector.
- From the command line
  - On Windows systems:

    start_*connName connName brokerName* [-c*configFile*]
  - On UNIX-based systems:

    connector_manager_*connName* -start

  where *connName* is the name of the connector and *brokerName* identifies your integration broker, as follows:
  - For WebSphere InterChange Server, specify for *brokerName* the name of the ICS instance.

– For WebSphere message brokers (WebSphere MQ Integrator, WebSphere MQ Integrator Broker, or WebSphere Business Integration Message Broker) or WebSphere Application Server, specify for *brokerName* a string that identifies the broker.

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

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

  You can load, activate, deactivate, pause, shutdown or delete a connector using this tool.
- From System Monitor (WebSphere InterChange Server product only)

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

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

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

Stopping the connector

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

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

    `connector_manager_`*`connName`* `-stop`

    where *connName* is the name of the connector.
- From Adapter Monitor (WebSphere Business Integration Adapters product only), which is launched when you start System Manager

  You can load, activate, deactivate, pause, shutdown or delete a connector using this tool.
- From System Monitor (WebSphere InterChange Server product only)

  You can load, activate, deactivate, pause, shutdown or delete a connector using this tool.
- On Windows systems, you can configure the connector to start as a Windows service. In this case, the connector stops when the Windows system shuts down.

Chapter 3. Understanding business objects for the connector

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

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

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

This chapter describes how the connector processes business objects and describes the assumptions the connector makes. You can use this information as suggestions for implementing new business objects or as a guide to modifying existing ones.

This chapter contains the following sections:

Business object and attribute naming conventions

For clarity, use the `PSFT_` prefix when you name an business object to represent data in a PeopleSoft Component Interface. Follow this convention for all child business objects as well as for the top-level business object. For example, if you create business objects that represent parent data in the `EMERGENCY_CNTCT` table and child data in the `EMERGENCY_PHONE` table, you might name the corresponding business objects: `PSFT_EmergencyContact` and `PSFT_EmergencyPhone`.

Note: In this document, the term **hierarchical** business object refers to a complete business object, including all the child business objects that it contains at any level. The term **individual** business object refers to a single business object, independent of any child business objects it might contain. The term

top-level business object refers to the individual business object at the top of the hierarchy that does not itself have a parent business object.

Business object structure

WebSphere business objects that represent PeopleSoft data can be flat or hierarchical. All the attributes of a **flat** business object are simple, that is, they represent a single value (such as a `String` or `Integer`).

In addition to its simple attributes, a **hierarchical** business object can have attributes that represent a single-cardinality child business object or an array of child business objects. In turn, each of these business objects can contain single-cardinality child business objects and arrays of business objects, and so on.

A **single-cardinality relationship** occurs when an attribute in a parent business object represents a single child business object. In this case, the child business object represents a PeopleSoft Collection that can contain only one Record. The type of the attribute is the same as that of the child business object.

A **multiple-cardinality relationship** occurs when an attribute in the parent business object represents an array of child business objects. In this case, the child business object represents a PeopleSoft Collection that can contain multiple Records. The type of the attribute is the same as the type of the array of child business objects.

The connector assumes that every hierarchical business object represents a single PeopleSoft Component Interface. Although a hierarchical business object may represent data in multiple PeopleSoft Records, the connector assumes that each child business object in the hierarchy represents a single Collection within the Component Interface. The connector uses PeopleSoft's Component architecture and processing (rather than the underlying database) to take advantage of the business logic defined at the Component level.

When you define a business object, the following situations are possible:

- The Collection might have more fields than the corresponding individual business object has simple attributes (that is, some fields in the Collection are not represented in the business object). Include in your design only those fields needed for the business object processing.
- The individual business object might have more simple attributes than the corresponding Collection has fields (that is, some attributes in the business object are not represented in the Collection). The attributes that do not have a representation in the Collection either have no application-specific information or are set with a default value.

Component interface and business object relationship

Because the connector requires a PeopleSoft Business Component and Component Interface for each business object that it processes, and because the WebSphere business integration system does not provide Business Components, Component Interfaces, or PeopleSoft-specific business objects, you must create these objects to use the connector.

Note: For portability, keep all your connector development in one PeopleSoft Project.

After creating the required objects mentioned above, you can use Application Designer to create the class structure required for the connector to process its supported business objects. For more information, see "Generating APIs" on page 35.

To assist you in creating these objects, this section includes:

- "Example Component and Component Interface" on page 31
- "Example business object" on page 34
- "Creating a business object" on page 34

Example Component and Component Interface

Figure 6 illustrates a simple Component, EMER_CONTACT, which has two Pages and three Records. The illustrated page, PERSONAL_DATA_PANEL1, stores emergency contact information for each employee.

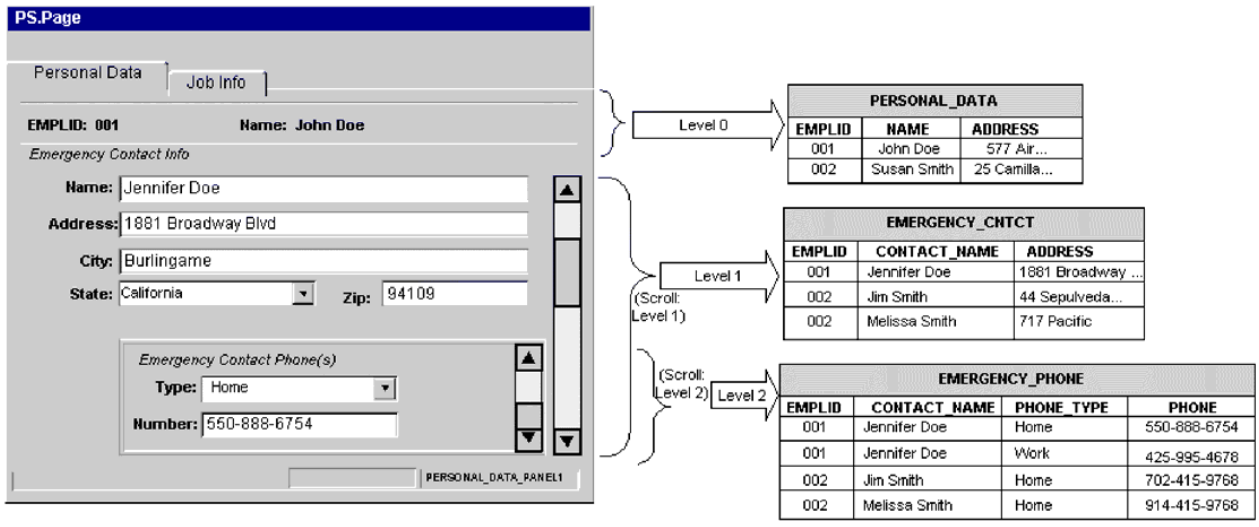

Figure 6. Example component

The above graphic illustrates the relationship between data displayed at each level of the personal data Page and the corresponding Record that stores it:

- Level 0 data is stored in the PERSONAL_DATA Record, whose key field is EMPLID.
- Level 1 data is stored in the EMERGENCY_CNTCT Record. Because each employee can have more than one emergency contact, each contact is uniquely identified by name (CONTACT_NAME), as well as by the employee's ID (EMPLID). The Scroll in Level 1 allows you to display data in the entire contact Collection.
- Level 2 data is stored in the EMERGENCY_PHONE Record. Because each contact can have more than one phone, each phone number is uniquely identified by type (PHONE_TYPE), as well as by the contact's name (CONTACT_NAME) and the employee's ID (EMPLID). The Scroll in Level 2 allows you to display data in the entire phone Collection.

As the EMERGENCY_CNTCT and EMERGENCY_PHONE Records illustrate, each Record's key is a composite that includes its parent's key as well as its own unique identifier.

Not all Components represent data records in the same straightforward manner as the example. For instance, scroll levels do not always correspond to separate Records or child Records. Sometimes a Component uses derived or work Records; sometimes it embeds related display Records; sometimes it uses the same Record

to represent data on several of its levels. In these situations, the design of the Component Interface is much more sophisticated than the example, and requires serious consideration of the processing needs.

When working with a complex Component, consider the following areas when designing:

- "Levels"
- "Hidden Fields"
- "Read-Only Fields"

Levels: When designing a Component's levels, consider the following:

- Verify that the Component Interface behaves like the online Pages contained in the Component that it represents. You may need to modify the structure of Component Interface properties and Collections, or add user-defined methods, to ensure that the connector behaves as designed.
- Keys are exposed only at the first level in which they appear. Remove keys from the top-level Collection of the Component Interface that do not appear at the level-0 Scroll in the page. Manually add those keys that appear on the level-1 Scroll, in the Page definition, to the level-two Collection.

  For example, assume you have a Page that uses three keys from the same Record (such as SETID, DEPTID, and EFFDT). Assume further that this Page uses EFFDT at Scroll Level 1 to return historical data for the given SETID and DEPTID. When you create a Component Interface that contains this Page, it displays all three keys in Collection Level 0 because all key fields exist in the same Level-0 primary Record. If you want to use this Component Interface to return a set of rows with EFFDT as the key, manually remove EFFDT from the Level-0 Collection and add it to the Level-1 Collection. Doing so causes the Component Interface to behave the same way as it does online.
- Examine the Page definitions of each Component, paying particular attention to the order of elements on a Page. Verify the Scroll level at which each field appears. Use this information to determine whether corresponding business object attributes are at the correct level, or whether they belong in a child business object.

Hidden Fields: Hidden fields are not always loaded into the Component Processor. Exposing these fields might cause application service errors. It is not recommended that you expose anything to the connector that is not visually exposed on the Page.

Read-Only Fields: Fields marked as Read-Only on the Component Interface are accessible to the connector, but only by the method that returns values. The connector cannot set values in these fields. Therefore, specify the `get=`*`FieldName`* parameter in the corresponding attribute's application specific text, but leave the `set=`*`FieldName`* parameter empty. Specifying the set parameter in such a case causes an error if the method is triggered by an update or create operation.

For more information, see "Application-specific information at the attribute level" on page 49.

Figure 7 illustrates the example Component Interface (`EMER_CONTACT_PROFILE`) that contains the `EMER_CONTACT` Component.

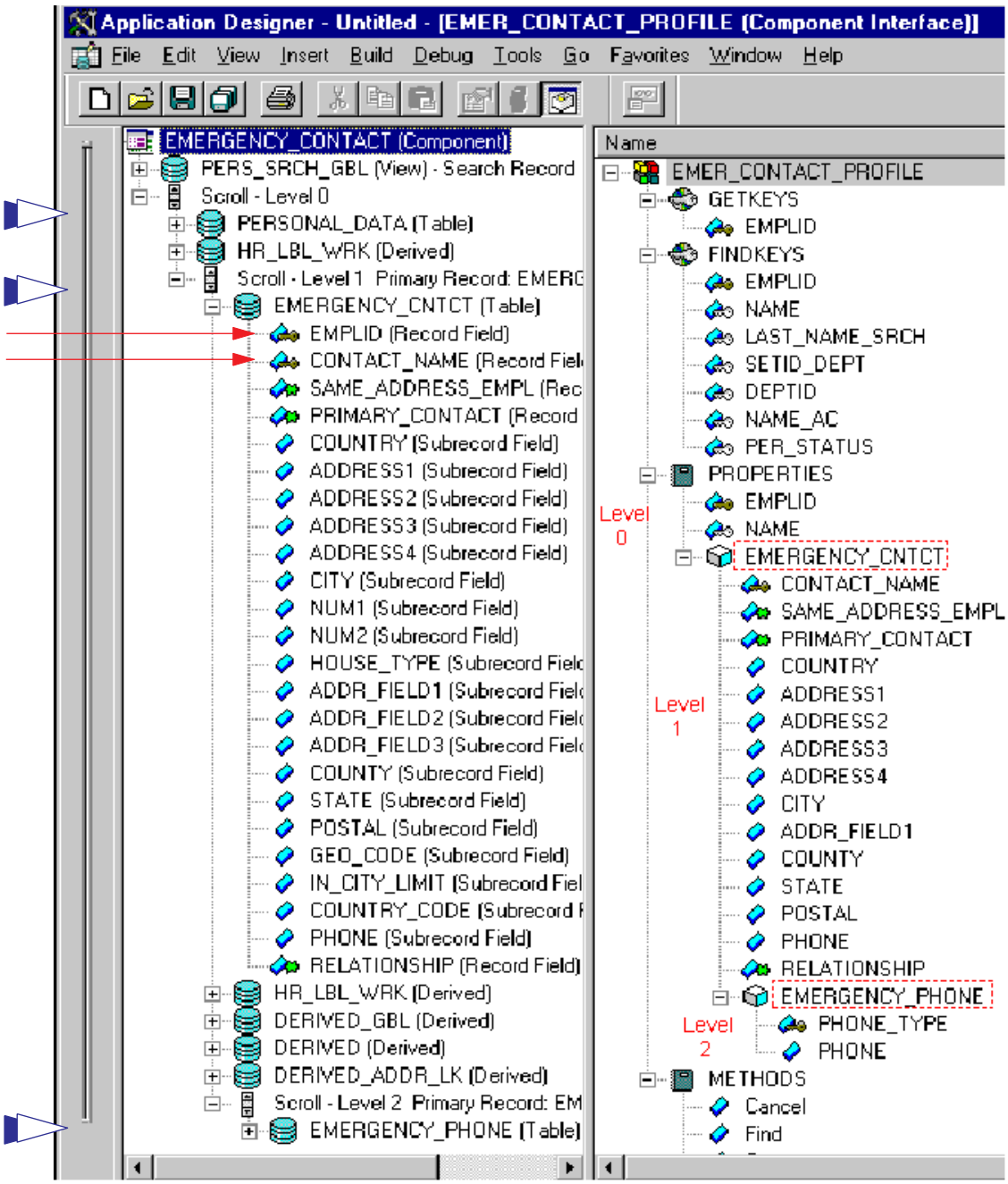

Figure 7. Example component interface

The three broad arrows in the left margin point to the tables (`PERSONAL_DATA`, `EMERGENCY_CNTCT`, and `EMERGENCY_PHONE`) that store data for the Component Interface. The two narrow arrows point to the key icons that indicate the key fields (`EMPLID` and `CONTACT_NAME`) of the `EMERGENCY_CNTCT` table.

PeopleSoft designed Component Interfaces to expose desired elements of a Component to third parties, making the business process logic (such as the PeopleCode, Field Edits, and PeopleSoft Security) transparent to external integration applications. Therefore, the connector relies on the application for any processing between the Component Interface and the database.

This dependency causes certain limitations, such as the lack of Search dialog processing. Consequently, the SearchInit, SearchSave, and RowSelect events are never triggered, and any PeopleCode associated with them does not run. This limitation also extends to any PeopleCode events that are exclusively related to GUI or online processing, including Menu PeopleCode and pop-up menus.

Therefore, before you create a Component Interface for the connector, verify that no important behavior will be lost, and that all predetermined data is in place before the connector accesses it. You can implement some of this skipped behavior as user-defined methods in the Component Interface or as Component-specific PeopleCode (such as the Pre-Build event). Failure to take these precautions can cause runtime errors.

To enable the connector to process the data represented by the EMER_CONTACT_PROFILE Component Interface, you must create PeopleSoft-specific business objects in the WebSphere business integration system.

Example business object

Figure 8 illustrates a hierarchical PeopleSoft-specific business object that you can create in the WebSphere business integration system to represent the example Component Interface.

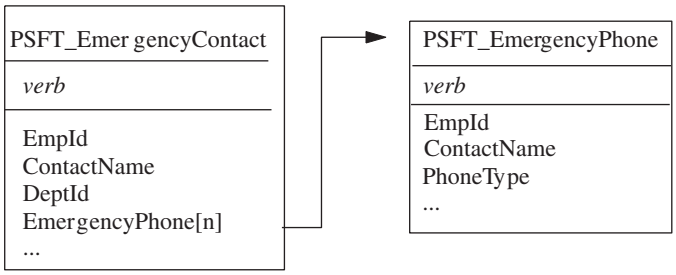

Figure 8. Example business objects

The PSFT_EmergencyContact business object contains two simple attributes that represent the key field data: EmpId and ContactName. These attributes correspond to EMPLID and CONTACT_NAME in the EMERGENCY_CNTCT Record. PSFT_EmergencyContact also has an array attribute (EmergencyPhone[n]) that represents the array of PSFT_EmergencyPhone business objects.

The PSFT_EmergencyPhone business object contains three simple attributes that represent the key field data: EmpId, ContactName, and PhoneType. The first two attributes serve as keys, uniquely identifying the parent business object. The third attribute uniquely distinguishes the child from other business objects in the same array.

Creating a business object

To create a supported business object, perform the following steps:

1. Locate the Component that represents the event and transaction. If necessary, create one.
2. Use the PeopleSoft Application Designer to create a Component Interface from the Component in step 1.
3. Use PeopleSoftODA or Business Object Designer to create a corresponding PeopleSoft-specific business object. For more information about business objects, see “Business object attribute properties” on page 45 and “Business

object application-specific information" on page 47. For more information about PeopleSoftODA, see Chapter 4, "Generating business object definitions using PeopleSoftODA," on page 53.

4. Add the PeopleSoft-specific business object to the list of the connector's supported business objects. For business-object subscription information specific to your integration broker, see the broker's implementation guide.

Use the PeopleSoft Application Designer to create a class structure from a Component Interface. The business object that corresponds to the Component Interface requires these classes to process the data; it is part of the PeopleSoft API.

To assist you in creating and using the generated classes, this section covers:

- "Generating APIs" on page 35
- "Example APIs" on page 36

Generating APIs

To generate APIs from a Component Interface, do the following:

1. Open the Component Interface in the Application Designer.
2. Select the `PeopleSoft APIs` menu from the `Build` menu.
3. Select only the Java option, specifying the destination as:

UNIX

`$ProductDirS/connectors/PeopleSoft/dependencies`

End of UNIX

Windows

`%ProductDir%\connectors\PeopleSoft\dependencies`

End of Windows

4. Compile the generated classes and verify that they are in the following directories:

UNIX

`$ProductDir directory:`

Component Interface class files

`/connectors/PeopleSoft/dependencies/PeopleSoft/Generated/CompIntfc`

Session-specific class files

`/connectors/PeopleSoft/dependencies/PeopleSoft/Generated/PeopleSoft`

End of UNIX

Windows

`%ProductDirS% directory:`

Component Interface class files

`\connectors\PeopleSoft\dependencies\PeopleSoft\Generated\CompIntfc`

Session-specific class files

```
\connectors\PeopleSoft\dependencies\PeopleSoft\Generated\PeopleSoft
```

End of Windows

For a more detailed explanation of the API-generating process, including screen shots, see "Building the API files" on page 18.

Example APIs

If you use the PeopleSoft APIs menu (a submenu of the Build menu) to create a class structure from the example EMER_CONTACT_PROFILE Component Interface, you have the following:

- EmerContactProfile.class—corresponds to the Component Interface

  The connector uses the name of this class to retrieve and instantiate the Component Interface in PeopleSoft. The classname is stored in the CiName property in the application-specific information at the business object level. For more information, see "Application-specific information at the business object level" on page 48.
- EmerContactProfileEmergencyCntct.class—corresponds to the EMERGENCY_CNTCT Collection.
- EmerContactProfileEmergencyCntctEmergencyPhone.class—corresponds to the EMERGENCY_PHONE Collection.

The generated methods that the connector uses are:

- "getFieldName() Method" on page 36
- "setFieldName() Method" on page 37
- "getCollectionName() Method" on page 37
- "CurrentItem() Method" on page 37
- "Item(index) Method" on page 37

getFieldName() Method

Each of the generated classes contains the get*FieldName*() method, which enables the connector to get the data value of each simple field of the Component Interface, and load it into the corresponding business object attribute.

For example, as illustrated in the right half of Figure 7, there are seven fields listed as FINDKEYS for the EMER_CONTACT_PROFILE Component. These fields include EMPLID, NAME, and DEPTID. To get data from these fields, the connector uses the getEmpId(), getName(), and getDeptId() methods. After obtaining the values, the connector loads them into the EmpId, ContactName, and DeptId business object attributes.

To return the value of simple fields in a Collection, the connector first returns the Collection, and then returns the fields in it. For example, to get the values in the CONTACT_NAME and SAME_ADDRESS_EMPL fields of the EMERGENCY_CNTCT Collection, the connector first executes the getEmergencyCntct() method. Then it executes the getContactName() and getSameAddressEmpl() methods. For more information, see "getCollectionName() Method" on page 37.

setFieldName() Method

Each of the generated classes contains the set*FieldName*() method, which enables the connector to set the data value of each simple field of the Component Interface based on the value of its corresponding business object attribute.

For example, to load data from the EmpId, ContactName, and DeptId business object attributes into the EMPLID, NAME, and DEPTID fields, the connector uses the setEmpId(), setName(), and setDeptId() methods.

getCollectionName() Method

To return the Collection of emergency contacts for a given employee, the connector uses the getEmergencyCntct() method in the EmerContactProfile class. The way that the connector handles the multiple rows depends on the setting of application-specific information at the business-object level.

- If the EFFDT parameter evaluates to true, the connector uses the CurrentItem() method to return only the record with the latest effective date.
- If the EFFDT parameter evaluates to false, the connector uses the Item(*index*) method to return only the first record, regardless of the effective date. For example, the connector returns a future date, if one is listed.

For information about the EFFDT parameter, see "Application-specific information at the business object level" on page 48.

To return the Collection of phone types and phone numbers for each of an employee's emergency contacts, the connector uses the getEmergencyPhone() method in the EmerContactProfileEmergencyCntct class. After obtaining all Records, the connector loads them into the business object's array attribute, EmergencyPhone[n].

CurrentItem() Method

When retrieving the records in a Collection, the connector uses the CurrentItem() method in the EmerContactProfile class to return only the record with the latest effective date. The connector uses this method only if the EFFDT parameter of the business-object level application-specific information evaluates to true.

Item(index) Method

When retrieving the records in a Collection, to return only the record with the specified record number, the connector uses the Item(*index*) method in the EmerContactProfile class. The connector uses this method only if the EFFDT parameter of the business-object level application-specific information evaluates to false. By default, this method returns the first row retrieved.

Business object verb processing

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

- "Afterimages and deltas" on page 38, which defines the terms and explains how the connector works with afterimages.
- "Verb Processing for business object requests" on page 39, which explains the steps the connector takes when creating, retrieving, updating, or deleting a business object.
- "Committing data" on page 45, which briefly explains how the connector saves data.

Afterimages and deltas

An afterimage is the state of a business object after all changes have been made to it. A delta is a business object used in an update operation that contains only key values and the data to be changed. This connector supports only afterimages, not business object deltas. When the connector receives a request business object for update, it assumes that the business object represents the desired state of the data after update.

Therefore, when the connector receives a request business object with the Update verb, it changes the current representation of the business object in the Component Interface so that it exactly matches the source business object. To do this, the connector changes simple attribute values and adds or removes child business objects.

For an example of how the connector modifies child business objects, assume that the `PSFT_EmergencyContact` business object has two additional attributes, one of which represents a single-cardinality child and the other of which represents an array of child business objects. Each child of the array can contain its own array of child business objects.

Figure 9 illustrates the current state of `PSFT_EmergencyContact` for an employee whose ID is `2345`. The `ArrayData` attribute represents three Records (A, B, and C). The array attribute in two of these records represents two additional Records.

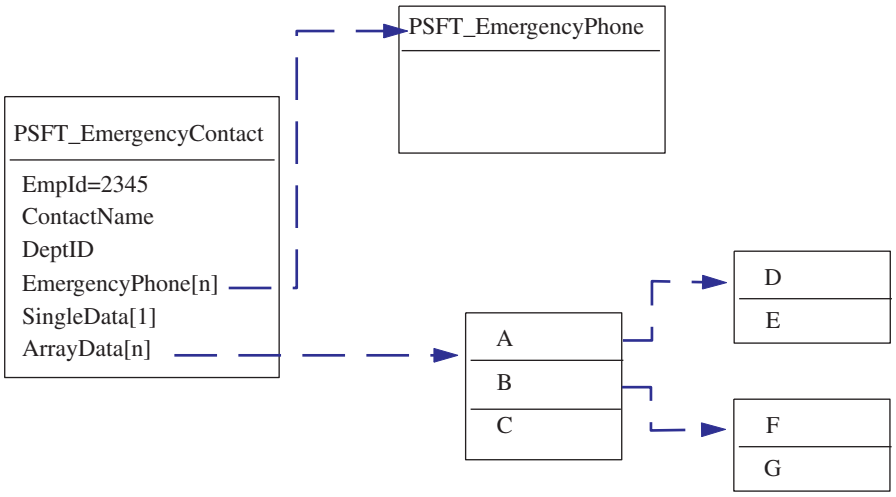

Figure 9. State of data prior to update

Figure 10 illustrates a business object request. This business object contains a new single-cardinality child business object and contains different business objects in its arrays.

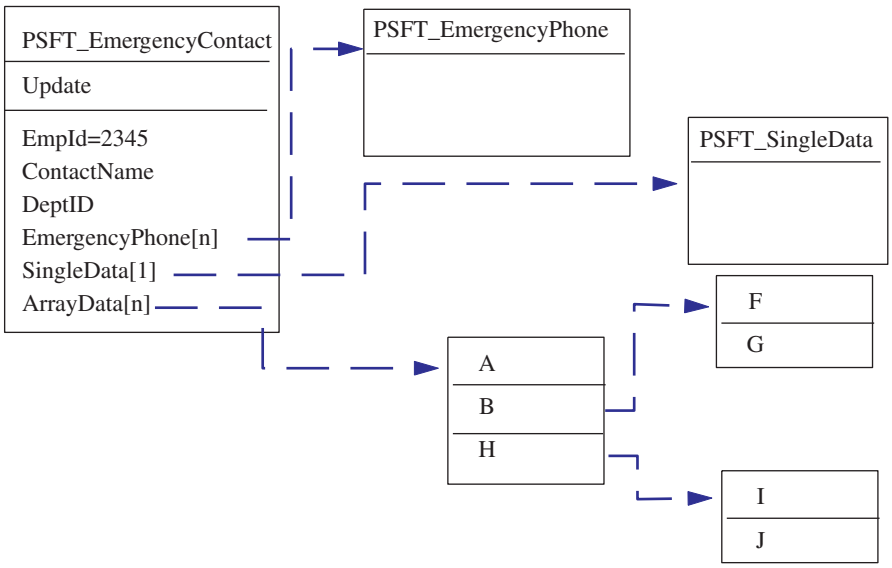

Figure 10. Data represented by an update request

To process the update, the connector applies the following changes to the Component Interface:

- Updates the simple attributes in the `PSFT_EmergencyContact` and the `PSFT_EmergencyPhone` business objects.
- Creates the `PSFT_SingleData` business object.
- Updates the simple attributes in the child business objects A, B, F and G.
- Deletes the child business objects C, D and E.
- Creates the child business objects H, I and J.

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

There is a way, however, that you can prevent the connector from deleting missing child business objects during an update operation. To instruct the connector to keep child business objects that are not included in the source business object, use the application-specific information for the attribute that represents the child or array of children. To do so, set `KeepRelationship` to `true`. For more information, see "Application-specific information at the attribute level" on page 49.

Verb Processing for business object requests

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

Business object comparison

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

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

Create operations

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

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

1. Creates a new instance of a Component Interface for the top-level business object.
   - Uses PeopleSoft's `Set`*`FieldName`*`(`*`value`*`)` method to populate properties in the Component Interface with values from attributes in the business object.

     Important: If the `UID` parameter of a key attribute's application-specific information evaluates to `false`, the business process that creates the business object must provide a new unique ID value for the attribute. If the business object does not have the required value, the connector logs an error.

     If the `UID` parameter of a key attribute's application-specific information evaluates to `true`, the application is responsible for generating a unique ID. In this case, the attribute's value or the `Default Value` property must contain the string `NEXT`. In other words, if the business process that creates the business object does not populate its value as `NEXT`, this value must be specified in the attribute's `Default Value` property. If the business object uses the Default Value property to provide the String `NEXT`, the connector's `UseDefaults` property must evaluate to `true`. For more information, see "UseDefaults" on page 24.

     If a unique identifier is not provided for the attribute's value, and the string `NEXT` is not specified in the `Default Value` property, the application logs a duplicate-key error.
   - Verifies that all Required attributes contain a value, and throws an error if such a value is missing. For more information on Required attributes, see "Required property" on page 46.
2. Recursively inserts each child business object and each array of child business objects into the Component Interface. In other words, the connector creates the child and all child business objects that the child and its children contain.

Note: If the business object definition for an attribute that represents a single-cardinality child business object specifies that the child is required (that is, its `Required` property evaluates to `true`), the retrieval must return a row. If it does not return a row, the connector returns an error and stops processing. However, if the child is not required and the attribute is empty, the connector ignores the attribute.

3. The connector calls the Save() method, which writes and commits the data.
   - If the connector is generating the unique ID, the generation occurs as this method executes.
   - If the application is generating the unique ID, the connector retrieves the key values set by the application after this method executes.

     Note: If InterChange Server (ICS) is the integration broker, the ID must be delivered synchronously because ICS requires the ID to cross-reference the business object.
   - If `setInteractiveMode` is defined as `false` in the business object's application-specific information, all PeopleCode editing occurs at this point. Any PeopleCode errors are published to the PSMessage collection queue.
   - If an instance of the Component Interface already exists with the same key values, the application returns a duplicate-key error, and the connector sends the `FAIL` return code.

For more information on attribute properties, see "Business object attribute properties" on page 45. For more information on specifying application-specific information, see "Application-specific information at the attribute level" on page 49.

Retrieve operations

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

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

1. Removes all child business objects from the top-level business object that it received from the integration broker.
2. Retrieves the Component Interface that corresponds to the top-level business object.

   The connector uses the key values in the source business object to instantiate the Component Interface. The result of the retrieval causes one of the following actions:
   - If it finds a Component Interface instance, the connector continues processing.
   - If it does not find a Component Interface instance, indicating that the top-level business object does not have a corresponding Component Interface in the application, the connector returns `FAIL`.
   - If it finds multiple Component Interface instances, the connector returns `MULTIPLE_HITS`.

   Note: A business object can contain attributes that do not correspond to any Component Interface property. During retrieval, the connector does not change such attributes in the top-level business object; they remain set to

the values it received. For child business objects, the connector sets such attributes to their default values during retrieval.

3. Recursively retrieves all Collections of the Component Interface that correspond to business object arrays.

   The connector uses the keys in each parent business object and the unique key of each child to select a data row from the Component Interface instance or Component Interface Collection. For each row returned, the connector performs the following actions:

   a. Creates a new individual business object of the correct type.
   b. Sets all of the current business object's attributes based on the values in the returned row.
   c. Recursively retrieves all of the current business object's children.
   d. Inserts the current business object with all of its children into the appropriate array of the parent.

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

4. Recursively retrieves the Collections for each of the top-level business object's single-cardinality children. The connector uses the keys in each parent business object and the unique key of each child to select a data row from the Component Interface instance or Component Interface Collection. The connector performs the following:

   a. If the business object's definition specifies that the child is required, the retrieval must return a row. If the child is not required and the retrieval returns no rows, indicating that the child does not exist in the Component Interface, the connector leaves the parent's single-cardinality attribute empty. If the retrieval returns more than one row, the retrieval fails.
   b. Recursively retrieves the Collections for all children contained by the child business object.
   c. Inserts the business object with all of its children into the appropriate attribute in the parent business object.

RetrieveByContent operations

When retrieving a business object, the connector returns a status of `VALCHANGE` if the operation was successful (regardless of whether the operation caused changes to the business object), `FAIL` if the operation failed, or `MULTIPLE_HITS` if the operation returned more than one row.

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

1. Removes all child business objects from the top-level business object that it received from the integration broker.
2. Retrieves the Component Interface that corresponds to the top-level business object.

   The connector uses the values of those attributes that are find keys in the source business object to instantiate the Component Interface. (For more information about specifying application-specific information for find key attributes, see "Application-specific information at the attribute level" on page 49.) The result of the retrieval causes one of the following actions:

   - If it finds a Component Interface instance, the connector continues processing.

- If it does not find a Component Interface instance, indicating that the top-level business object does not have a corresponding Component Interface in the application, the connector returns `FAIL`.
- If it finds multiple Component Interface instances, the connector returns `MULTIPLE_HITS`.

Note: A business object can contain attributes that do not correspond to any Component Interface property. During retrieval, the connector does not change such attributes in the top-level business object; they remain set to the values it received. For child business objects, the connector sets such attributes to their default values during retrieval.

3. Recursively retrieves all Collections of the Component Interface that correspond to business object arrays.

   The connector uses the keys in each parent business object and the unique key of each child to select a data row from the Component Interface instance or Component Interface Collection. For each row returned, the connector performs the following actions:

   a. Creates a new individual business object of the correct type.
   b. Sets all of the current business object's attributes based on the values in the returned row.
   c. Recursively retrieves all of the current business object's children.
   d. Inserts the current business object with all of its children into the appropriate array of the parent.

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

4. Recursively retrieves the Collections for each of the top-level business object's single-cardinality children. The connector uses the keys in each parent business object and the unique key of each child to select a data row from the Component Interface instance or Component Interface Collection. The connector performs the following:

   a. If the business object's definition specifies that the child is required, the retrieval must return a row. If the child is not required and the retrieval returns no rows, indicating that the child does not exist in the Component Interface, the connector leaves the parent's single-cardinality attribute empty. If the retrieval returns more than one row, the retrieval fails.
   b. Recursively retrieves the Collections for all children contained by the child business object.
   c. Inserts the business object with all of its children into the appropriate attribute in the parent business object.

Update operations

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

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

1. Uses the key values of the source business object to retrieve the corresponding Component Interface instance. The retrieved Component Interface is an accurate representation of the current state of the data in the PeopleSoft application.
   - If the retrieval fails, indicating that the top-level business object does not exist in the application, the connector returns `BO_DOES_NOT_EXIST`.
   - If the retrieval succeeds, the connector compares the retrieved Component Interface to the source business object to determine which child business objects require changes in the Component Interface. The connector does not, however, compare values in the source business object's simple attributes to those in the retrieved Component Interface; the connector updates the value of all simple attributes.
2. Recursively updates all single-cardinality children of the top-level business object.

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

   The connector processes the update of single-cardinality child business objects in one of the following ways:
   - If the child is present in both the source business object and the retrieved Component Interface, the connector recursively updates it in the Component Interface.

     Note: The source business object and the retrieved Component Interface must match. If the two hierarchical objects contain the same single-cardinality ownership children in different order, the connector returns an error and stops processing.
   - If the child is present in the source business object but not in the retrieved Component Interface, the connector recursively creates it in the Component Interface.

     Important: If the `UID` parameter of a key attribute's application-specific information evaluates to `false`, the business process that creates the business object must provide a new unique ID value for the attribute. If the business object does not have the required value, the connector logs an error.

     If the `UID` parameter of a key attribute's application-specific information evaluates to `true`, the application is responsible for generating a unique ID. In this case, the attribute's value or the `Default Value` property must contain the string `NEXT`. In other words, if the business process that creates the business object does not populate its value as `NEXT`, this value must be specified in the attribute's `Default Value` property. If the business object uses the Default Value property to provide the String `NEXT`, the connector's `UseDefaults` property must evaluate to `true`. For more information, see "UseDefaults" on page 24.

     If a unique identifier is not provided for the attribute's value, and the string `NEXT` is not specified in the `Default Value` property, the application logs a duplicate-key error.
   - If the child is present in the retrieved Component Interface but not in the source business object, the connector recursively deletes it from the

Component Interface. However, if the `KeepRelationship` parameter of the parent's application-specific information evaluates to `true`, the connector preserves the child business object.

When deleting child business objects during an Update operation, the connector uses PeopleSoft's `deleteItem()` method to delete the corresponding Collection from the Component Interface instance. The connector physically or logically deletes only Collections that are at level 1 or higher.

For more information on specifying application-specific information, see "Application-specific information at the attribute level" on page 49.

3. Updates all simple attributes of the retrieved Component Interface except those whose corresponding attribute in the source business object contain the value `CxIgnore`.
4. Processes all arrays of the retrieved Component Interface in one of the following ways:
   - If a child exists in both the source business object's array and the retrieved Component Interface's array, the connector recursively updates it in the Component Interface.
   - If a child exists in the source array but not in the retrieved Component Interface's array, the connector recursively creates it in the Component Interface.
   - If the child exists in the retrieved Component Interface's array but not in the source array, the connector recursively deletes it from the Component Interface.

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

Note: The connector locks data while retrieving it to ensure data integrity.

Delete operations

The connector does not delete a top-level business object. However, it does physically delete a child business object if its top-level business object uses the Update verb, and the child does not exist in the request business object that represents source data.

For more information, see "Update operations" on page 43.

Committing data

Whenever the connector receives a business object for create or update processing, it either saves all changes to the Component Interface or none. The connector never saves subset of data changes.

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.

Name property

Each business object attribute must have a unique name.

Type property

Each business object attribute must have a type, such as `Integer`, `String` or the type of a child business object.

Cardinality property

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

Key property

At least one simple attribute of each business object must be specified as the key. To do so, set this 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 connector uses each key attribute to uniquely identify or create an instance of a Component Interface. For information about causing the application to generate a unique ID, see "Create operations" on page 40 and "Update operations" on page 43.

Required property

The `Required` property specifies whether a simple attribute or an attribute that represents a single-cardinality child business object must contain a value.

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

When the connector receives a business object with a Create request, the connector causes the Create operation to fail if a required attribute does not have a valid value.

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

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

Max length property

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

AppSpecificInfo

For information on this property, see "Application-specific information at the attribute level" on page 49.

Default value property

This property specifies a default value that the connector uses to populate a simple field if the attribute does not contain a value. The connector does not evaluate this property for attributes that represent child business objects. For a create operation, the connector uses the value of this property only if its `UseDefaults` property evaluates to `true`. For more information, see "UseDefaults" on page 24.

The connector sends the value of this property to the application to use in identifier-generation if the following is true:

- The `Default Value` property is specified for a key attribute whose application-specific information specifies `true` in its `UID` parameter
- The value of the `Default Value` property is the string `NEXT`
- The connector's `UseDefaults` property evaluates to `true`

For information about causing the application to generate a unique ID, see "Create operations" on page 40 and "Update operations" on page 43.

Special attribute value

Simple attributes in business object can have a special value: `CxIgnore`. When the connector receives a request business object, the connector ignores all attributes with a value of `CxIgnore`. It is as if those attributes were invisible to the connector.

When the connector retrieves data from a Component Interface with a field that contains a null value, the connector sets the value of its corresponding attribute to `CxIgnore` by default.

Because the connector requires at least one key attribute to create a business object, the business process that creates the business object should ensure that business objects passed to the connector have at least one key that is not set to `CxIgnore`. The only exception to this requirement is a business object whose key is to be generated by the connector.

Business object application-specific information

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

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

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

This section provides information on the application-specific information format for business objects supported by the connector.

Table 7 provides an overview of the functionality available in business object application-specific information.

Table 7. Overview of application-specific information in supported business objects

Scope of application-specific information	Functionality
Entire business object	Specifies: • The name of the corresponding Component Interface • Whether triggering of online Field PeopleCode edits is immediate or batch • Whether all rows associated with component interface are retrieved or only the most recent instance
Simple attributes	Specifies: • Whether the application generates unique identifier values or whether the connector provides them • Whether an attribute should be used as a key for retrieving data
Business object verb	The connector does not specify functionality based on the verb.

The following sections discuss this functionality in more detail.

Application-specific information at the business object level

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

- Specify the name of the corresponding Component Interface.
- Define whether online Field PeopleCode triggers immediately after update or during `save()` method
- Indicate whether or not all rows from a Component Interface are retrieved regardless of effective date.

At the business-object level, application-specific information format consists of four parameters separated by a colon (:) delimiter. The format is:

```
cIName=<ComponentInterface>
:EFFDT=[true|false]:setInteractiveMode=[true|false]:GetHistoryItems
=[true|false]:setEditHistoryItems=[true|false]
:GetDummyRows=[true|false]:InsAtOldestEffDtPos=[true]
:InsAtCurrentEffDtPos=[true]
```

Table 8 describes these parameters.

Table 8. AppSpecificInfo at the business-object level

AppSpecificinfo parameter	Description
`cIName`	Specifies the name of the Component Interface defined in the PeopleSoft application.
`EFFDT (isEffectiveDated)`	Specifies whether the business object (or child business object) uses an Effective Date: • If the `EFFDT` parameter evaluates to `true`, the connector uses the `CurrentItem()` method to return only the record with the latest effective date. • If the `EFFDT` parameter evaluates to `false`, the connector uses the `Item(index)` method to return only the first record, regardless of the effective date. For example, the connector returns a future date, if one is listed. The default value is `true`.

Table 8. AppSpecificInfo at the business-object level (continued)

AppSpecificinfo parameter	Description
setInteractiveMode	Determines when the connector sends changes to the Application Server. This property is typically used to enhance performance. • When set to true, the connector processes each time the value of a property or attribute changes; in other words, all online Field PeopleCode immediately triggers after the connector calls set*PropertyName*() on each attribute. • When set to false, the PeopleSoft application batches the processing, sending changes to the Application Server only when the connector calls Save() on a Component Interface. The default value is true.
GetHistoryItems	Determines how much data the connector retrieves. • When set to true, the connector retrieves all data rows, regardless of effective date, for the corresponding Component Interface. • When set to false, the connector retrieves only the current data (the effective row). The default value is true.
GetDummyRows	Supports the GetDummyRowsproperty in PeopleTools 8.4 and higher. Do not set this if you are using older versions of PeopleTools. For information about this property, see your PeopleSoft documentation.
InsAtOldestEffDtPos	This parameter is provided to address limitations in older versions of PeopleTools where the call getEffectiveItemNum()returns an insert position of -1 for effective-dated rows. When InsAtOldestEffDtPos is set, the connector inserts the row at the highest index, which has the oldest effective date. If you set this parameter, do not set InsAtCurrentEffDtPos.
InsAtCurrentEffDtPos	This parameter is provided to address limitations in older versions of PeopleTools where the call getEffectiveItemNum()returns an insert position of -1 for effective-dated rows. When InsAtCurrentEffDtPos is set, the connector inserts the row at the lowest index zero, which has the current effective date. If you set this parameter, do not set InsAtOldestEffDtPos.
SetEditHistoryItems	Enables editing and saving of history data. Applies to effective-dated fields only.

For more information about the setInteractiveMode and GetHistoryItems properties, see the documentation from PeopleSoft.

For example, the PSFT_EmergencyContact business object might have the following value specified for its business object application-specific information:

```
cINAME=EMER_CONTACT_PROFILE:setInteractiveMode=false:
GetHistoryItems=true:isEffectiveDated=false
```

Application-specific information at the attribute level

Application-specific information at the attribute level specifies the connector's behavior on an attribute-by-attribute level. The format of the application-specific

information is a set of five name-value parameters, each of which includes the parameter name and its value. Each parameter set is separated from the next by a colon (:) delimiter. A vertical bar (|) separates the members of a set of options. The format is:

```
get=getFieldName:set=setFieldName:UID=[true|false]:GetKey=[true|false]:KeepRelationship
=[true|false]:findKey=[true|false]:bigDec=true:EFFDT=[true|false]
:EFFDTSEQ=[true|false]
```

Important: Case is significant when the connector evaluates application-specific information.

For example, for a Read-Only simple attribute that should be used in addition to the key fields when retrieving data, you might specify the following format:

```
get=getBusinessUnit:GetKey=true
```

For a multiple-cardinality attribute whose children should be preserved even if they are not included in an Update business object request, you might specify the following format:

```
get=getEmergencyPhone:set=setEmergencyPhone:KeepRelationship=true
```

Table 9 describes each name-value parameter.

Table 9. Name-value parameters in attribute application-specific information

Parameter	Description
get=get*FieldName*	This parameter specifies the method to use when retrieving a value from a Component Interface field or Component Interface Collection to the current attribute.
set=set*FieldName*	This parameter specifies the method to use when setting a value in a Component Interface field or Component Interface Collection, based on the value in the current attribute.
UID=[true|false]	If this parameter is set to `false`, a unique value for the attribute must be provided by the business process that creates the business object. If the attribute does not contain a value, the connector sends an error message. If this parameter is set to `true`, the application is responsible for generating a unique ID (using auto-numbering). Identifier generation requires the attribute's value to contain the String `NEXT`. The business process that creates the business object must populate its value as `NEXT`, or this value can be specified in the attribute's `Default Value` property. If the business object uses the Default Value property to provide the String `NEXT`, the connector's `UseDefaults` property must evaluate to `true`. For more information, see "UseDefaults" on page 24. If the key value is set to anything other than varchar, and you want the application to generate the IDs, leave the value blank and UID=True. Only set the value to NEXT when the key value is set to a varchar datatype. When the application generates the unique ID, the connector retrieves the generated ID from the application after the Save operation completes and the data has been committed. When ICS is the integration broker, the connector uses the retrieved ID in cross-referencing the business object. If an attribute does not require the application to generate a unique ID, set this value to `false`, or do not include this parameter in the application-specific information.

Table 9. Name-value parameters in attribute application-specific information (continued)

Parameter	Description
GetKey=[true | false]	If this parameter is set to `true` and the attribute's Key and Required properties evaluate to `true`, the connector includes the attribute as part of the key during a retrieve or retrieve-for-update operation. The connector uses this parameter to differentiate a retrieval key from a create key when a distinction is necessary. When executing a create operation, the connector provides as keys only those attributes whose Key and Required properties evaluate to `true`. Because the create-key fields alone do not always uniquely retrieve required data, the connector adds to the keys all attributes whose GetKey parameter evaluates to `true` when it executes a retrieve operation.
KeepRelationship=[true | false]	Used only on an attribute that represents an array of child business objects, this parameter specifies whether the connector deletes existing child business objects that are not represented in the source business object during an update operation. • Set to `true` to prevent deletion. • Set to `false` to allow deletion. For example, assume an existing phone number is associated with an existing contact. Assume further that the connector receives a request to update a PSFT_EmergencyContact business object that contains a single child business object. The child associates an emergency phone number with the contact. If `KeepRelationship` evaluates to `true` for the `EmergencyPhone[n]` attribute, the connector updates the contact by adding its new association without deleting its existing association. However, if `KeepRelationship` evaluates to `false`, the connector deletes all existing child data that is not contained in the source business object. In such a case, the contact is associated only with the new phone number.
findKey=[true | false]	If this parameter is set to `true`, the connector includes the attribute as a key during a retrieve-by-content operation. Set this parameter for attributes that are find keys in the Component Interface.
bigDec=[true | false]	This parameter identifies bigdecimal type attributes. Set this parameter to `true` for bigdecimal attributes in the Component Interface.
EFFDT=[true | false]	This parameter identifies effective-dated attributes in the Component Interface. The connector only uses this if, at the business object level, application-specific information has `EFFDT` set to true.
EFFDTSEQ=[true | false]	This parameter identifies the sequence of effective-dated attributes based on which insert position getEffectiveItemNum returns. The connector only uses this if the business object definition includes an effective-dated attribute.

Chapter 4. Generating business object definitions using PeopleSoftODA

This chapter describes PeopleSoftODA, an object discovery agent (ODA), that generates business object definitions for the IBM WebSphere Business Integration Adapter for PeopleSoft. Because the connector works with objects that are based on PeopleSoft Component Interfaces and their associated Collections, PeopleSoftODA uses the Component Interface Java API to discover business object requirements specific to the PeopleSoft data source.

This chapter contains the following sections:

Installation and usage

This section discusses the following:

Installing PeopleSoftODA

To install PeopleSoftODA, use Installer for IBM WebSphere Business Integration Adapters. Follow the instructions in the *Implementation Guide for MQ Integrator*, or, for InterChange System (ICS), the *System Installation Guide for UNIX* or for Windows. When the installation is complete, the following files are installed in the product directory on your system:

- `ODA\PeopleSoft\PeopleSoftODA.jar`
- `ODA\messages\PeopleSoftODAAgent.txt`
- ODA\messages\PeopleSoftODAAgent_*ll_TT*.txt files (message files specific to a language (_*ll*) and a country or territory (_*TT*))
- `ODA\PeopleSoft\start_PeopleSoftODA.bat` (Windows only)
- `ODA/PeopleSoft/start_PeopleSoftODA.sh` (UNIX only)

Note: Except as otherwise noted, this document uses backslashes (\) as the convention for directory paths. For UNIX installations, substitute slashes (/) for backslashes. All product pathnames are relative to the directory where product is installed on your system.

Before using PeopleSoftODA

Before you run PeopleSoftODA, verify that your system has the required files and that the variables are correctly set in the script or batch file that runs the ODA.

This chapter assumes that you have already followed the instructions for installing the connector. As part of the connector's installation, you should have followed instructions in the following sections:

- "Required software for using the connector" on page 12—for information on downloading `psjoa.jar`
- "Enabling the application for the connector" on page 15—for information on creating `PSFTCI.jar`

Therefore, you should find the following files in the `connectors\PeopleSoft\dependencies` directory below the product directory:

- `psjoa.jar`—Downloaded from `\web\PSJOA` in the `PS_HOME` directory. PeopleSoftODA uses this file to synchronously send Component Interfaces and their information through the Jolt portion of the PeopleSoft Application Server.
- `PSFTCI.jar`—Created from Component Interface definitions in the Application Designer. You must compile the Component Interface API files after you generate them in PeopleSoft. PeopleSoftODA uses this file to generate business object definitions. For more information, see "Generating APIs" on page 35.

Important: It is recommended that you regenerate and recompile all component interfaces prior to running PeopleSoftODA to assure consistency. If the component interfaces are not in a jar file, or if either of the above jar files are not in the correct directory, modify the start script or batch file to locate them.

Open for editing the shell or batch file and confirm that the values described in Table 10 are correct.

Table 10. Shell and batch file configuration variables

Variable	Explanation	Example
`set AGENTNAME`	Name of the ODA	`set AGENTNAME = PeopleSoftODA`
`set AGENT`	Name of the ODA's jar file	**UNIX:** `set AGENT = ${`*`ProductDir`*`S}/ODA/PeopleSoft/PeopleSoftODA.jar` **WINDOWS:** `set AGENT = %`*`ProductDir`*`S%\ODA\PeopleSoft\PeopleSoftODA.jar`

Note: If you register PeopleSoftODA as a CORBA object or with an Object Activation Daemon (OAD), you can modify the class path for the PeopleSoft driver through the object discovery agent registration wizard. For information on registering the ODA, see the *System Installation Guide for Unix* or for Windows.

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

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

The following sections describe these steps in detail.

Launching PeopleSoftODA

Launch the PeopleSoftODA with the appropriate script:

•

UNIX

```
start_PeopleSoftODA.sh
```

End of UNIX

Windows

```
start_PeopleSoftODA.bat
```

End of Windows

You configure and run PeopleSoftODA using Business Object Designer. Business Object Designer locates each ODA by the name specified in the AGENTNAME variable of each script or batch file. The default ODA name for this connector is PeopleSoftODA. During installation, if you register the ODA with an Object Activation Daemon, the wizard automatically prefixes the hostname to the AGENTNAME value to make it unique.

Running multiple instances of PeopleSoftODA

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

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

It is recommended that you prefix each name with the name of the host machine when you run ODA instances on different machines. If you registered the ODA with an Object Activation Daemon, you can use an ORB finder (osfind) to locate existing CORBA object names on your network.

Figure 11 on page 57 illustrates the window in Business Object Designer from which you select the ODA to run.

Note: The connection properties must be the same for each particular ODA instance. Each additional ODA instance must to connect to application server instance to which it first connected.

Working with error and trace message files

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

```
AgentNameAgent.txt
```

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

- If you change the name of an ODA and do not create a message file for it, you must change the name of the message file in Business Object Designer as part of ODA configuration. Business Object Designer provides a name for the message

file but does not actually create the file. If the file displayed as part of ODA configuration does not exist, change the value to point to an existing file.

- You can copy the existing message file for a specific ODA, and modify it as required. Business Object Designer assumes you name each file according to the naming convention. For example, if the AGENTNAME variable specifies `PeopleSoftODA1`, the tool assumes that the name of the associated message file is `PeopleSoftODA1Agent.txt`. Therefore, when Business Object Designer provides the filename for verification as part of ODA configuration, the filename is based on the ODA name. Verify that the default message file is named correctly, and correct it as necessary.

Important: Failing to correctly specify the message file's name when you configure the ODA causes it to run without messages. For more information on specifying the message file name, see "Configure initialization properties" on page 57.

During the configuration process, you specify:

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

Table 11 describes these values.

Table 11. 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 PeopleSoftODA spawned • Traces the business object definition dump

For information on where you configure these values, see "Configure initialization properties" on page 57.

Using PeopleSoftODA in business object designer

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

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

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

1. Open Business Object Designer.
2. From the File menu, select the New Using ODA... submenu.

Business Object Designer displays the first window in the wizard, named Select Agent. Figure 11 on page 57 illustrates this window.

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

1. "Select the ODA"
2. "Configure initialization properties" on page 57
3. "Expand nodes and select component interfaces and collections" on page 59
4. "Confirm selection of objects" on page 60
5. "Generate the definition" on page 61 and, optionally, "Provide additional information" on page 61
6. "Save the definition" on page 62

Select the ODA

Figure 11 illustrates the first dialog box in Business Object Designer's six-step wizard. From this window, select the ODA to run.

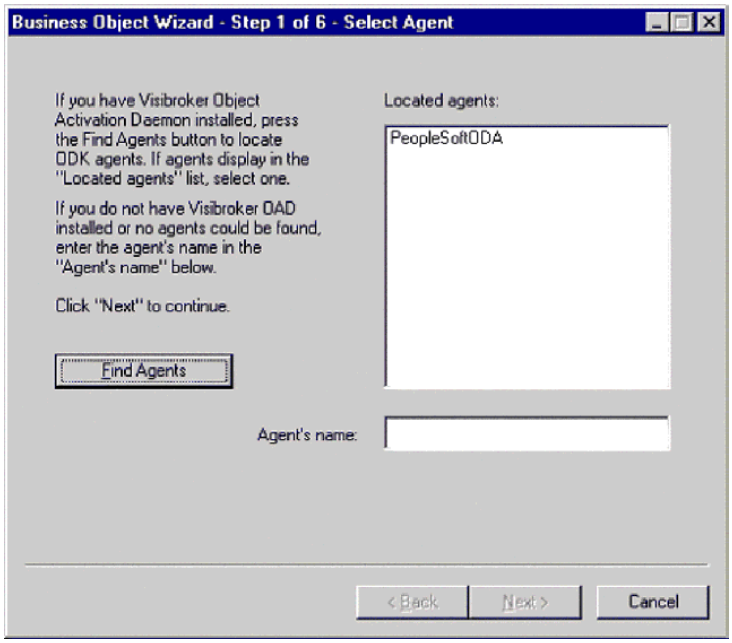

Figure 11. Selecting the ODA

To select the ODA:

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

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

2. Select the desired ODA from the displayed list.

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

Configure initialization properties

The first time Business Object Designer communicates with PeopleSoftODA, it prompts you to enter a set of initialization properties as shown in Figure 12. You can save these properties in a named profile so that you do not need to re-enter

them each time you use PeopleSoftODA. For information on specifying an ODA profile, see the *Business Object Development Guide*.

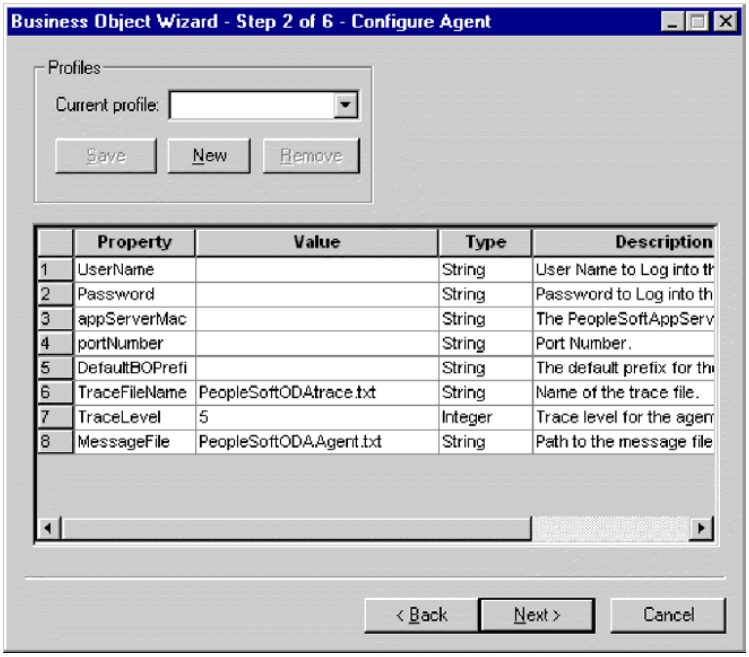

Figure 12. Configuring agent initialization properties

Configure the PeopleSoftODA properties described in Table 12.

Table 12. PeopleSoftODA properties

Row number	Property name	Property type	Description
1	`UserName`	String	Name of the user with authorization to connect to the PeopleSoft application using the Application Server
2	`Password`	String	Password of the user with authorization to connect to the PeopleSoft application
3	`AppServerMachineName`	String	Name or IP address of the machine on which the PeopleSoft Server is running
4	`PortNumber`	String	The Jolt port used to connect to the Application Server. The default value is `9000`. Note: This port is different from the Tuxedo port.
5	`DefaultBOPrefix`	String	Text that is prepended to the name of the business object to make it unique. The prefix cannot begin with an underscore (_). You can change this value later, if required, when Business Object Designer prompts you for business object properties. For more information, see “Provide additional information” on page 61.
6	`TraceFileName`	String	File into which PeopleSoftODA writes trace information. If the file does not exist, PeopleSoftODA creates it in the `\ODA\PeopleSoft` directory. If the file already exists, PeopleSoftODA appends to it. PeopleSoftODA names the file according to the naming convention. For example, if the agent is named `PeopleSoftODA`, it generates a trace file named `PeopleSoftODAtrace.txt`. Use this property to specify a different name for this file.

Table 12. PeopleSoftODA properties (continued)

Row number	Property name	Property type	Description
7	`TraceLevel`	Integer	Level of tracing enabled for PeopleSoftODA. For more information, see "Working with error and trace message files" on page 55.
8	`MessageFile`	String	Name of the error and message file. PeopleSoftODA displays the filename according to the naming convention. For example, if the agent is named `PeopleSoftODA`, the value of the message file property displays as `PeopleSoftODAAgent.txt`.**Important**: The error and message file must be located in the `\ODA\messages` directory. Use this property to verify or specify an existing file.

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

Expand nodes and select component interfaces and collections

Business Object Designer uses the properties configured in the previous step to create a connect string that connects the tool to the specified PeopleSoft application. After connecting, Business Object Designer displays a tree whose nodes represent all the Component Interfaces defined in the PeopleSoft application.

Click on a node to display the next-level Collection of the Component Interface. You can expand the Component Interfaces to display the entire hierarchical representation. For each Collection, PeopleSoftODA creates a child business object definition.

Figure 13 illustrates this dialog box with some Component Interfaces expanded.

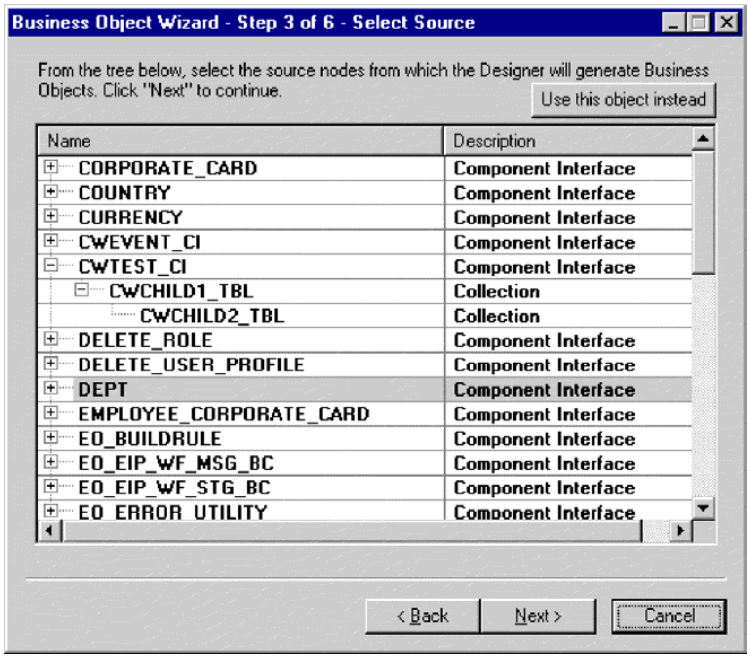

Figure 13. Tree of component interfaces with expanded nodes

Select all required Component Interfaces, along with all necessary Collections, and click Next.

Confirm selection of objects

After you identify all the Component Interfaces and Collections to be associated with the generated business object definitions, Business Object Designer displays the dialog box with only the selected objects. Figure 14 illustrates this dialog box.

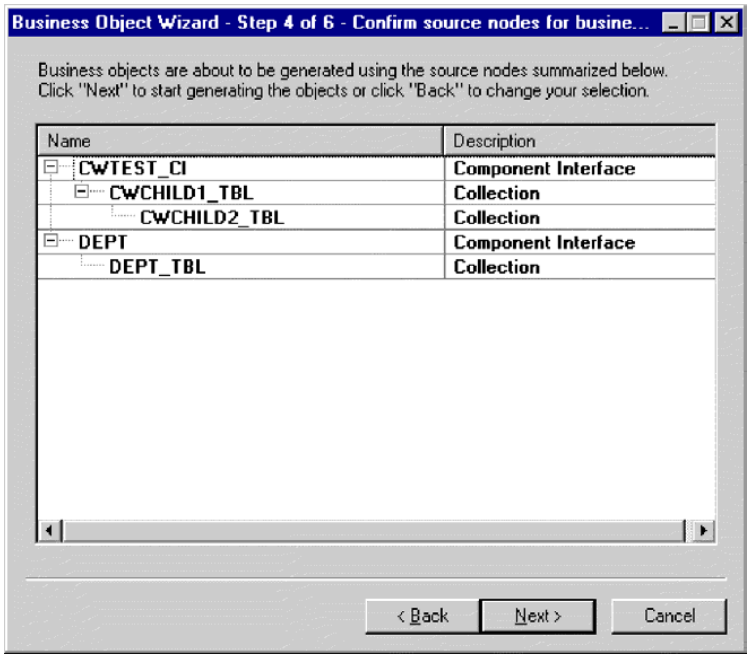

Figure 14. Confirming selection of objects

This window provides the following options:

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

Generate the definition

After you confirm the Component Interfaces and Collections, the next dialog box informs you that Business Object Designer is generating the definitions. If a large number of Component Interfaces has been selected, this generation step can take time.

Figure 15 illustrates this dialog box.

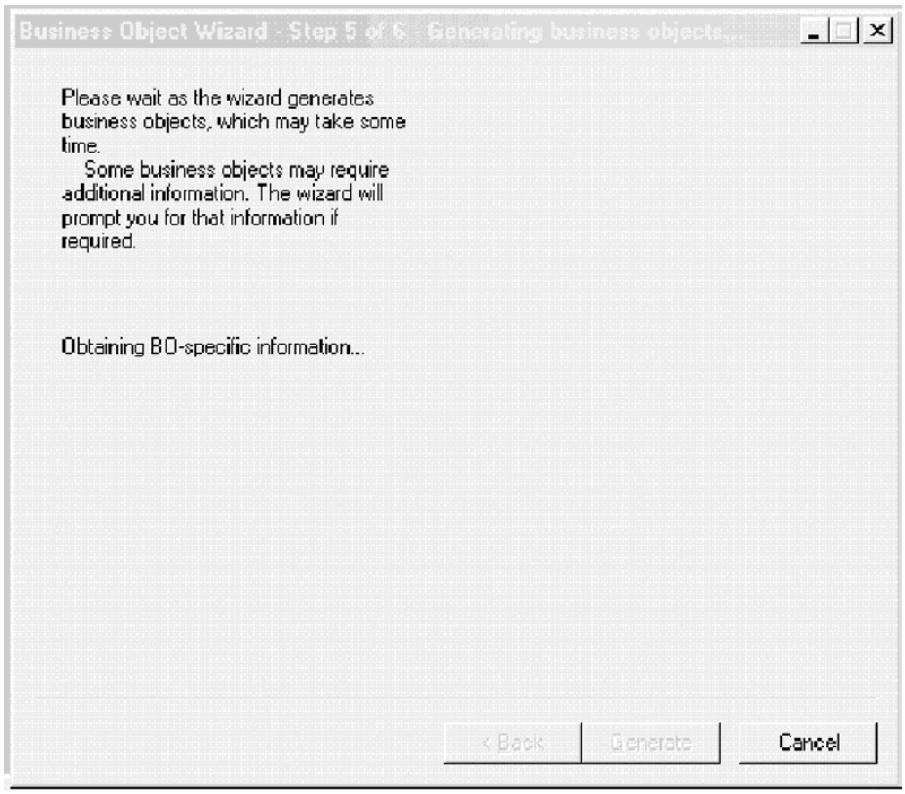

Figure 15. Generating the definition

Provide additional information

If the PeopleSoftODA needs additional information, Business Object Designer displays the BO Properties window, which prompts you for the information. Figure 16 illustrates this dialog box.

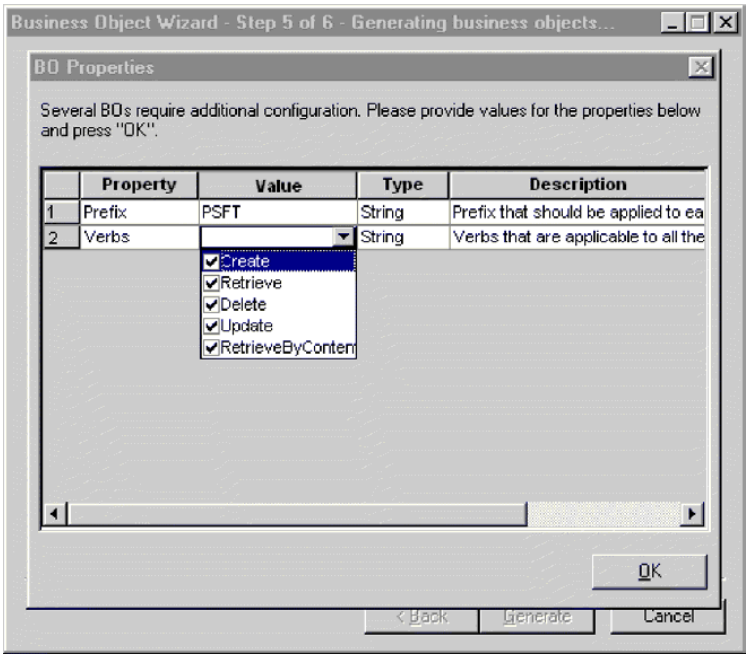

Figure 16. Providing additional Information

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

- *Prefix*—The text that is prepended to the name of the business object to make it unique. If you are satisfied with the value you entered for the *DefaultBOPrefix* property in the Configure Agent window (Figure 12 on page 58), 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.

Save the definition

After you provide all required information in the BO Properties dialog box and click OK, Business Object Designer displays the final dialog box in the wizard. In this dialog box, you can save the definition to the server (if ICS is the integration broker) or to a file (for any integration broker), or you can open the definition for editing in Business Object Designer. For more information, and to make further modifications, see the *Business Object Development Guide*.

Figure 17 illustrates this dialog box.

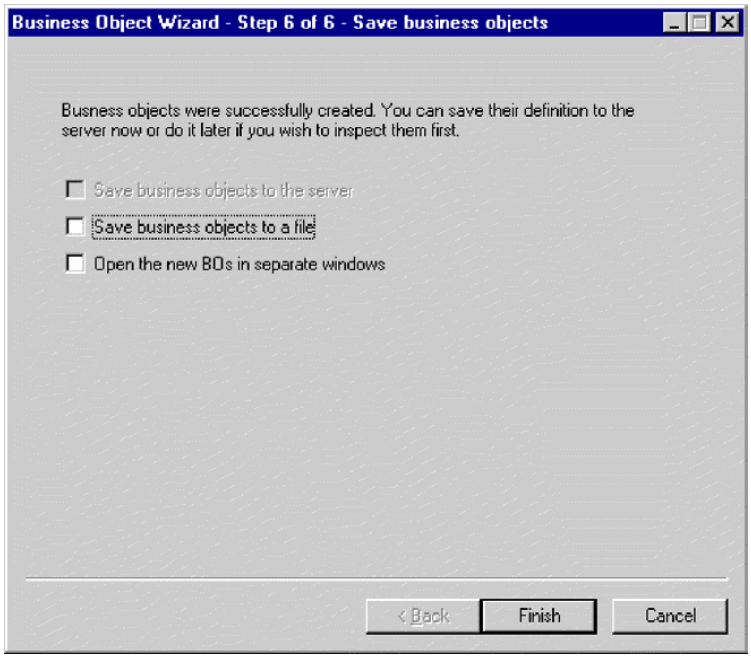

Figure 17. Saving the business object definition

Contents of the generated definition

The business object definition that PeopleSoftODA generates contains:

- A simple attribute for each property in the specified Component Interface
- An attribute that represents an array of child business objects for each Collection that was selected in the hierarchical representation of the Component Interface
- The verbs specified in the BO Properties window (Figure 16 on page 62)
- Application-specific information:
  - At the business-object level
  - For each attribute

This section describes:

- "Business-object-level properties"
- "Attribute properties" on page 64

Business-object-level properties

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

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

At the business-object level, application-specific information format consists of four parameters separated by a colon (:) delimiter. The format is:

```
cIName=ComponentInterface:EFFDT=[true|false]:setInteractiveMode=[true|false]:
GetHistoryItems=[true|false]:SetEditHistoryItems=[true|false]
```

Table 13 describes these parameters.

Table 13. AppSpecificInfo at the business-object level

AppSpecificInfo Parameter	Description
`ComponentInterface`	Specifies the name of the Component Interface defined in the PeopleSoft application
`setInteractiveMode`	Determines when the connector sends changes to the Application Server
`GetHistoryItems`	Determines how much data the connector retrieves
`EFFDT`	Specifies whether the business object (or child business object) uses an Effective Date
`SetEditHistoryItems`	Enables editing and saving of history data. Applies to effective-dated fields only.

Important: PeopleSoftODA uses the name of the Component Interface from which it has generated the definition to specify a value for the `ComponentInterface` property. It does not provide values for the other properties. You must modify the business object definition to provide values for the remaining properties. For information on these properties, see "Application-specific information at the business object level" on page 48. For information on modifying a business object definition, see the *Business Object Development Guide*.

Attribute properties

This section describes the properties that PeopleSoftODA generates for each attribute. For more information about the attributes, see "Business object attribute properties" on page 45.

Name property

PeopleSoftODA derives the attribute's name from a property in the corresponding Component Interface.

Data type property

When setting the type of an attribute, PeopleSoftODA converts the data type of a property into a corresponding business object data type as shown in Table 14:

Table 14. Correspondence of data types

PeopleSoft	Business object	Length
String	String	Length specified in the data type
Boolean	Boolean	
Collection	Object	
Float	Float	
Number	Integer	

Note: If a property's data type is not one of those shown in Table 14, PeopleSoftODA skips the property and displays a message stating that the property cannot be processed.

Cardinality property

PeopleSoftODA sets the cardinality of all simple attributes to 1. It sets the cardinality of all attributes that represent an array of child business objects to `n`.

MaxLength property

PeopleSoftODA provides the default length of 255 characters for strings; for all other data types, it uses the standard maximum length for the corresponding business object data type.

IsKey property

If a property is a `CreateKey` in a Component Interface, PeopleSoftODA sets this property to `true`. If a property is a `GetKey` in a Component Interface, PeopleSoftODA sets this property to `false`, and sets the attribute's AppSpecificInfo parameter to `GetKey=true`.

IsForeignKey property

PeopleSoftODA sets this property to `false`. You can change the setting in Business Object Designer.

IsRequired property

Because PeopleSoftODA generates some keys internally, it always sets this property to `false`. You can change the setting in Business Object Designer.

AppSpecificInfo property

Attribute application-specific information is a set of five name-value parameters that are separated from one another by a colon (:) delimiter. A vertical bar (|) separates the members of a set of options. The format is:

```
get=get<FieldName>:set=set<FieldName>:UID=[true|false]:GetKey=[true|false]:KeepRelationship=[true|false]
```

PeopleSoftODA generates only those properties that are relevant to an attribute, as described in Table 15. If it generates more than one parameter, it separates parameters with a colon.

Table 15. Attribute AppSpecificInfo generated by PeopleSoftODA

AppSpecificInfo parameter	Description
`GetKey=true`	PeopleSoftODA generates this parameter only for attributes that correspond to a Component Interface property defined as a `GetKey`. The connector uses the value of such an attribute to retrieve Component Interface instances.
`get=get`*`PropertyName`*	For `PropertyName`, PeopleSoftODA substitutes the name of the Component Interface property associated with the attribute. It generates this parameter for every simple attribute that corresponds to a Component Interface property. The connector uses this method to retrieve values for the attribute.
`get=get`*`CollectionName`*	For `CollectionName`, PeopleSoftODA substitutes the name of the Component Interface Collection associated with the attribute. It generates this parameter for every attribute that represents an array of child business objects corresponding to a Component Interface Collection. The connector uses this method to retrieve the Collection.
`set=set`*`PropertyName`*	For `PropertyName`, PeopleSoftODA substitutes the name of the Component Interface property associated with the attribute. It generates this parameter for every simple attribute that corresponds to a Component Interface property. The connector uses the method to update values for the attribute.

Note: You can set additional AppSpecificInfo parameters in Business Object Designer. For information about these parameters, see "Application-specific information at the attribute level" on page 49. For more information on modifying definitions, see the *Business Object Development Guide*. For an example of using the parameters described in Table 15, see "Sample business object definition file" on page 66.

Verbs

Note: PeopleSoftODA generates the verbs specified in the BO Properties window (as illustrated in Figure 16 on page 62).

Sample business object definition file

There are three sample business object definition files included with the product:

- BO_Psft_DEPT
- BO_PsftEmployee
- SavePostChange

BO_Psft_DEPT business object

The following example is the BO_Psft DEPT business object.

```
[BusinessObjectDefinition] Name = DeptTbl Version = 1.0.0 AppSpecificInfo =
CiName=DEPT     [Attribute]     Name = Company     Type = String
MaxLength = 255     IsKey = true     IsForeignKey = false     IsRequired =
false     AppSpecificInfo = get=getCompany:set=setCompany
IsRequiredServerBound = false     [End]     [Attribute]     Name =
BudgetLvl     Type = String     MaxLength = 1     IsKey = false
IsForeignKey = false     IsRequired = false     AppSpecificInfo =
get=getBudgetLvl:set=setBudgetLvl     IsRequiredServerBound = false
[End]     [Attribute]     Name = Descr     Type = String     MaxLength = 1
    IsKey = false     IsForeignKey = false     IsRequired = false
AppSpecificInfo = get=getDescr:set=setDescr     IsRequiredServerBound =
false     [End]     [Attribute]     Name = DescrShort     Type = String
MaxLength = 1     IsKey = false     IsForeignKey = false     IsRequired =
false     AppSpecificInfo = get=getDescrshort:set=setDescrshort
IsRequiredServerBound = false     [End]     [Attribute]     Name =
ObjectEventId     Type = String     MaxLength = 255     IsKey = false
IsForeignKey = false     IsRequired = false     IsRequiredServerBound =
false     [End]     [Verb]     Name = Create     [End]     [Verb]     Name
= Delete     [End]     [Verb]     Name = Retrieve     [End]     [Verb]
Name = Update     [End] [End] [BusinessObjectDefinition] Name = Psft_dept
Version = 1.0.0 AppSpecificInfo = CiName=DEPT     [Attribute]     Name =
Deptid     Type = String     MaxLength = 255     IsKey = true
IsForeignKey = false     IsRequired = true     AppSpecificInfo =
get=getDeptid:set=setDeptid:GetKey=true     IsRequiredServerBound = false
    [End]     [Attribute]     Name = Setid     Type = String     MaxLength
= 1     IsKey = true     IsForeignKey = false     IsRequired = true
AppSpecificInfo = get=getSetid:set=setSetid:GetKey=true
IsRequiredServerBound = false     [End]     [Attribute]     Name = DptTbl
    Type = DeptTbl     ContainedObjectVersion = 1.0.0     Relationship =
Containment     Cardinality = n     MaxLength = 1     IsKey = false
IsForeignKey = false     IsRequired = true     AppSpecificInfo =
get=getDeptTbl:KEEPRELATIONSHIP=true     IsRequiredServerBound = false
```

```
[End]     [Attribute]     Name = ObjectEventId     Type = String
MaxLength = 255     IsKey = false     IsForeignKey = false     IsRequired =
false     IsRequiredServerBound = false     [End]     [Verb]     Name =
Create     [End]     [Verb]     Name = Delete     [End]     [Verb]     Name
= Retrieve     [End]     [Verb]     Name = Update     [End] [End]
```

BO_PsftEmployee business object

The following example is the BO_PsftEmployee business object.

```
[BusinessObjectDefinition] Name = PSFTEmployee Version = 1.0.0
AppSpecificInfo = cIName=Emp     [Attribute]     Name = EMPID     Type =
String     Cardinality = 1     MaxLength = 255     IsKey = true
IsForeignKey = false     IsRequired = true     AppSpecificInfo =
get=getEmplid:set=setEmplid:keepRelationship=false:uid=true:
findKey=true:getKey=true:createKey=true     IsRequiredServerBound = false
    [End]     [Attribute]     Name = EMPL_RCD     Type = String
Cardinality = 1     MaxLength = 1     IsKey = true     IsForeignKey = false
    IsRequired = true     AppSpecificInfo =
get=getEmplRcd:set=setEmplRcd:keepRelationship=false:uid=true:
findKey=true:getKey=true:createKey=true     IsRequiredServerBound = false
    [End]     [Attribute]     Name = NAME     Type = String     Cardinality
= 1     MaxLength = 1     IsKey = true     IsForeignKey = false
IsRequired = true     AppSpecificInfo =
get=getName:set=setName:keepRelationship=false:uid=false:
findKey=true:getKey=false:createKey=false     IsRequiredServerBound = false
    [End]     [Attribute]     Name = LAST_NAME_SRCH     Type = String
Cardinality = 1 MaxLength = 1     IsKey = true     IsForeignKey = false
IsRequired = true     AppSpecificInfo =
get=getLastNameSrch:set=setLastNameSrch:keepRelationship=false:uid=false:
    findKey=true:getKey=false:createKey=false     IsRequiredServerBound =
false     [End]     [Attribute]     Name = NAME_AC     Type = String
Cardinality = 1     MaxLength = 1     IsKey = true     IsForeignKey = false
    IsRequired = true     AppSpecificInfo =
get=getNameAc:set=setNameAc:keepRelationship=false:uid=false:
findKey=true:getKey=false:createKey=false     IsRequiredServerBound = false
    [End]     [Attribute]     Name = PER_STATUS     Type = String
Cardinality = 1     MaxLength = 1     IsKey = true     IsForeignKey = false
    IsRequired = true     AppSpecificInfo =
get=getPerStatus:set=setPerStatus:keepRelationship=false:uid=false:
findKey=true:getKey=false:createKey=false     IsRequiredServerBound = false
    [End]     [Attribute]     Name = EMPLID_0     Type = String
Cardinality = 1     MaxLength = 1     IsKey = false     IsForeignKey =
false     IsRequired = false     AppSpecificInfo =
get=getEmplid0:set=setEmplid0:keepRelationship=false:uid=false:
findKey=false:getKey=false:createKey=false     IsRequiredServerBound =
false     [End]     [Attribute]     Name = ORIG_HIRE_DT     Type = String
    Cardinality = 1     MaxLength = 1     IsKey = false     IsForeignKey =
false     IsRequired = false     AppSpecificInfo =
get=getOrigHireDt:set=setOrigHireDt:keepRelationship=false:uid=false:
findKey=false:getKey=false:createKey=false     IsRequiredServerBound =
false     [End]     [Attribute]     Name = SEX     Type = String
Cardinality = 1     MaxLength = 1     IsKey = false     IsForeignKey =
false     IsRequired = true     AppSpecificInfo =
get=getSex:set=setSex:keepRelationship=false:uid=false:findKey=false:
getKey=false:createKey=false     IsRequiredServerBound = false     [End]
```

```
    [Attribute]     Name = BIRTHDATE     Type = String     Cardinality = 1
    MaxLength = 1     IsKey = false     IsForeignKey = false     IsRequired
= false     AppSpecificInfo =
get=getBirthdate:set=setBirthdate:keepRelationship=false:
uid=false:findKey=false:getKey=false:createKey=false
IsRequiredServerBound = false     [End]     [Attribute]     Name =
FT_STUDENT     Type = String     Cardinality = 1     MaxLength = 1
IsKey = false     IsForeignKey = false     IsRequired = true
AppSpecificInfo = get=getFtStudent:set=setFtStudent:keepRelationship=false:
    uid=false:findKey=false:getKey=false:createKey=false
IsRequiredServerBound = false     [End]     [Attribute]     Name =
BENEFIT_RCD_NBR     Type = String     Cardinality = 1     MaxLength = 1
IsKey = false     IsForeignKey = false     IsRequired = false
AppSpecificInfo = get=getBenefitRcdNbr:set=setBenefitRcdNbr:
keepRelationship=false:uid=false:findKey=false:getKey=false:createKey=false
    IsRequiredServerBound = false     [End]     [Attribute]     Name =
HOME_HOST_CLASS     Type = String     Cardinality = 1     MaxLength = 1
IsKey = false     IsForeignKey = false     IsRequired = true
AppSpecificInfo = get=getHomeHostClass:set=setHomeHostClass:
keepRelationship=false:uid=false:findKey=false:getKey=false:createKey=false
    IsRequiredServerBound = false     [End]     [Attribute]     Name =
HIRE_DT     Type = String     Cardinality = 1     MaxLength = 1     IsKey =
false     IsForeignKey = false     IsRequired = false     AppSpecificInfo =
get=getHireDt:set=setHireDt:keepRelationship=false:
uid=false:findKey=false:getKey=false:createKey=false
IsRequiredServerBound = false     [End]     [Attribute]     Name =
CMPNY_SENIORITY_DT     Type = String     Cardinality = 1     MaxLength = 1
    IsKey = false     IsForeignKey = false     IsRequired = false
AppSpecificInfo = get=getCmpnySeniorityDt:
set=setCmpnySeniorityDt:keepRelationship=false:uid=false:
findKey=false:getKey=false:createKey=false     IsRequiredServerBound =
false     [End]     [Attribute]     Name = SERVICE_DT     Type = String
Cardinality = 1     MaxLength = 1     IsKey = false     IsForeignKey =
false     IsRequired = false     AppSpecificInfo =
get=getServiceDt:set=setServiceDt:
keepRelationship=false:uid=false:findKey=false:getKey=false:createKey=false
    IsRequiredServerBound = false     [End]     [Attribute]     Name =
PROF_EXPERIENCE_DT     Type = String     Cardinality = 1     MaxLength = 1
    IsKey = false     IsForeignKey = false     IsRequired = false
AppSpecificInfo = get=getProfExperienceDt:
set=setProfExperienceDt:keepRelationship=false:uid=false:
findKey=false:getKey=false:createKey=false     IsRequiredServerBound =
false     [End]     [Attribute]     Name = LAST_VERIFICATN_DT     Type =
String     Cardinality = 1     MaxLength = 1     IsKey = false
IsForeignKey = false     IsRequired = false     AppSpecificInfo =
get=getLastVerificatnDt:set=setLastVerificatnDt:
keepRelationship=false:uid=false:findKey=false:getKey=false:createKey=false
    IsRequiredServerBound = false     [End]     [Attribute]     Name =
EXPECTED_RETURN_DT     Type = String     Cardinality = 1     MaxLength = 1
    IsKey = false     IsForeignKey = false     IsRequired = false
AppSpecificInfo = get=getExpectedReturnDt:set=
setExpectedReturnDt:keepRelationship=false:
uid=false:findKey=false:getKey=false:createKey=false
IsRequiredServerBound = false     [End]     [Attribute]     Name =
LAST_DATE_WORKED     Type = String     Cardinality = 1     MaxLength = 1
    IsKey = false     IsForeignKey = false     IsRequired = false
```

```
AppSpecificInfo = get=getLastDateWorked:
set=setLastDateWorked:keepRelationship=false:uid=false:
findKey=false:getKey=false:createKey=false     IsRequiredServerBound =
false     [End]     [Attribute]     Name = LAST_INCREASE_DT     Type =
String     Cardinality = 1     MaxLength = 1     IsKey = false
IsForeignKey = false     IsRequired = false     AppSpecificInfo =
get=getLastIncreaseDt:     set=setLastIncreaseDt:keepRelationship=false:
    uid=false:findKey=false:getKey=false:createKey=false
IsRequiredServerBound = false     [End]     [Attribute]     Name =
OWN_5PERCENT_CO     Type = String     Cardinality = 1     MaxLength = 1
IsKey = false     IsForeignKey = false     IsRequired = true
AppSpecificInfo = get=getOwn5percentCo:
set=setOwn5percentCo:keepRelationship=false:
uid=false:findKey=false:getKey=false:createKey=false
IsRequiredServerBound = false     [End]     [Attribute]     Name =
BUSINESS_TITLE     Type = String     Cardinality = 1     MaxLength = 1
IsKey = true     IsForeignKey = false     IsRequired = true
AppSpecificInfo = get=getBusinessTitle:
set=setBusinessTitle:keepRelationship=false:
uid=false:findKey=false:getKey=true:createKey=false
IsRequiredServerBound = false     [End]     [Attribute]     Name =
REPORTS_TO     Type = String     Cardinality = 1     MaxLength = 1
IsKey = false     IsForeignKey = false     IsRequired = false
AppSpecificInfo = get=getReportsTo:set=setReportsTo:
keepRelationship=false:uid=false:findKey=false:getKey=false:createKey=false
    IsRequiredServerBound = false     [End]     [Attribute]     Name =
SUPERVISOR_ID     Type = String     Cardinality = 1     MaxLength = 1
IsKey = false     IsForeignKey = false     IsRequired = false
AppSpecificInfo = get=getSupervisorId:set=setSupervisorId:
keepRelationship=false:uid=false:findKey=false:getKey=false:createKey=false
    IsRequiredServerBound = false     [End]     [Attribute]     Name =
PROBATION_DT     Type = String     Cardinality = 1     MaxLength = 1
IsKey = false     IsForeignKey = false     IsRequired = false
AppSpecificInfo = get=getProbationDt:set=setProbationDt:
keepRelationship=false:uid=false:findKey=false:getKey=false:createKey=false
    IsRequiredServerBound = false     [End]     [Attribute]     Name =
SECURITY_CLEARANCE     Type = String     Cardinality = 1     MaxLength = 1
    IsKey = false     IsForeignKey = false     IsRequired = false
AppSpecificInfo = get=getSecurityClearance:
set=setSecurityClearance:keepRelationship=false:
uid=false:findKey=false:getKey=false:createKey=false
IsRequiredServerBound = false     [End]     [Attribute]     Name = PHONE
    Type = String     Cardinality = 1     MaxLength = 1     IsKey = false
    IsForeignKey = false     IsRequired = false     AppSpecificInfo =
get=getPhone:set=setPhone:     keepRelationship=false:uid=false:
findKey=false:getKey=false:createKey=false     IsRequiredServerBound =
false     [End]     [Attribute]     Name = TIME_RPT_LOCK     Type = String
    Cardinality = 1     MaxLength = 1     IsKey = false     IsForeignKey =
false     IsRequired = false     AppSpecificInfo =
get=getTimeRptLock:set=setTimeRptLock:
keepRelationship=false:uid=false:findKey=false:getKey=false:createKey=false
    IsRequiredServerBound = false     [End]     [Attribute]     Name =
JOB_REPORTING     Type = String     Cardinality = 1     MaxLength = 1
IsKey = false     IsForeignKey = false     IsRequired = true
AppSpecificInfo = get=getJobReporting:set=setJobReporting:
keepRelationship=false:uid=false:findKey=false:getKey=false:createKey=false
```

```
    IsRequiredServerBound = false     [End]     [Attribute]     Name =
DED_TAKEN     Type = String     Cardinality = 1     MaxLength = 1     IsKey
= false     IsForeignKey = false     IsRequired = true     AppSpecificInfo
= get=getDedTaken:set=setDedTaken:
keepRelationship=false:uid=false:findKey=false:getKey=false:createKey=false
    IsRequiredServerBound = false     [End]     [Attribute]     Name =
DED_SUBSET_ID     Type = String     Cardinality = 1     MaxLength = 1
IsKey = false     IsForeignKey = false     IsRequired = false
AppSpecificInfo = get=getDedSubsetId:set=setDedSubsetId:
keepRelationship=false:uid=false:findKey=false:getKey=false:createKey=false
    IsRequiredServerBound = false     [End]     [Attribute]     Name =
CAN_ABORIGINAL     Type = String     Cardinality = 1     MaxLength = 1
IsKey = false     IsForeignKey = false     IsRequired = false
AppSpecificInfo = get=getCanAboriginal:set=setCanAboriginal:
keepRelationship=false:uid=false:findKey=false:getKey=false:createKey=false
    IsRequiredServerBound = false     [End]     [Attribute]     Name =
CAN_VISBL_MINORITY     Type = String     Cardinality = 1     MaxLength = 1
    IsKey = false     IsForeignKey = false     IsRequired = true
AppSpecificInfo = get=getCanVisblMinority:set=setCanVisblMinority:
keepRelationship=false:uid=false:findKey=false:getKey=false:createKey=false
    IsRequiredServerBound = false     [End]     [Attribute]     Name =
CURRENT_SEQ     Type = String     Cardinality = 1     MaxLength = 1
IsKey = false     IsForeignKey = false     IsRequired = false
AppSpecificInfo = get=getCurrentSeq:set=setCurrentSeq:
keepRelationship=false:uid=false:findKey=false:getKey=false:createKey=false
    IsRequiredServerBound = false     [End]     [Attribute]     Name =
PERS_DATA_EFFDT     Type = PERS_DATA_EFFDT     ContainedObjectVersion =
1.0.0     Relationship = Containment     Cardinality = n     MaxLength = 1
    IsKey = false     IsForeignKey = false     IsRequired = false
AppSpecificInfo = get=getPersDataEffdt:keepRelationship=false:
uid=false:findKey=false:getKey=false:createKey=false
IsRequiredServerBound = false     [End]     [Attribute]     Name =
EMAIL_ADDRESSES     Type = EMAIL_ADDRESSES     ContainedObjectVersion =
1.0.0     Relationship = Containment     Cardinality = n     MaxLength = 1
    IsKey = false     IsForeignKey = false     IsRequired = false
AppSpecificInfo = get=getEmailAddresses:keepRelationship=false:
uid=false:findKey=false:getKey=false:createKey=false
IsRequiredServerBound = false     [End]     [Attribute]     Name =
PERSONAL_PHONE     Type = PERSONAL_PHONE     ContainedObjectVersion = 1.0.0
    Relationship = Containment     Cardinality = n     MaxLength = 1
IsKey = false     IsForeignKey = false     IsRequired = false
AppSpecificInfo = get=getPersonalPhone:keepRelationship=false:
uid=false:findKey=false:getKey=false:createKey=false
IsRequiredServerBound = false     [End]     [Attribute]     Name = PERS_NID
    Type = PERS_NID     ContainedObjectVersion = 1.0.0     Relationship =
Containment     Cardinality = n     MaxLength = 1     IsKey = false
IsForeignKey = false     IsRequired = false     IsRequiredServerBound =
false     [End]     [Attribute]     Name = JOB     Type = JOB
ContainedObjectVersion = 1.0.0     Relationship = Containment
Cardinality = n     MaxLength = 1     IsKey = false     IsForeignKey =
false     IsRequired = false     IsRequiredServerBound = false     [End]
    [Attribute]     Name = ObjectEventId     Type = String     MaxLength =
255     IsKey = false     IsForeignKey = false     IsRequired = false
IsRequiredServerBound = false     [End]     [Verb]     Name = Create
[End]     [Verb]     Name = Delete     [End]     [Verb]     Name = Retrieve
    [End]     [Verb]     Name = Update     [End] [End]
```

SavePostChange business object

The following example is the SavePostChange business object.

```
/* Place this code in Component's SavePostChg() and define the four
parameters used in the function call */ Declare Function cw_publish_event
PeopleCode FUNCLIB_CW.CW_EVENT_NOT FieldFormula; Component string &BONAME1;
Component string &KEYLIST1; Component number &CWPRIORITY1; Component string
&CONNID; &BONAME1 = "Psft_Dept"; &KEYLIST1 =
"DEPT_TBL.SETID:DEPT_TBL.DEPTID"; &CWPRIORITY1 = 2; &CONNID = "PeopleSoft
Connector"; /*  Check if Component Changed before calling function */ If
ComponentChanged() And       %UserId <> "CW" Then /* Publish this event to
the CrossWorlds  CW_EVENT_TBL for polling */     cw_publish_event(&BONAME1,
&KEYLIST1, &CWPRIORITY1, &CONNID);    End-If;
```

Modifying information in the business object definition

It may be necessary modify information in the business object definition that PeopleSoftODA creates. For example, you must manually remove unwanted attributes, change the default values for the `getHistoryItems` and `setInteractiveMode` parameters of the application-specific information at the business-object level, and add required parameters for attribute application-specific information. For more information, see Chapter 3, “Understanding business objects for the connector,” on page 29.

To examine or modify the business object definition, you can use Business Object Designer or a text editor. To reload a revised definition into the repository, you can use Business Object Designer. Alternatively, if WebSphere InterChange Server (ICS) is the integration broker, you can use the `repos_copy` command to load the definition into the repository; if WebSphere MQ Integrator broker is the integration broker, you can use a system command to copy the file into the repository directory.

Chapter 5. Troubleshooting and error handling

The chapter describes problems that you may encounter when starting up or running the IBM WebSphere Business Integration Adapter for PeopleSoft.

This chapter contains the following sections:

Startup problems

If you encounter difficulties when trying to start the connector, do the following:

- Use the `psadmin` utility to check that the PeopleSoft Application Server is running.
- Use the `Test Component Interface` utility in Application Designer to verify that the Component Interface is working online.
- Rebuild and recompile the APIs in PeopleSoft. For more information, see "Generating APIs" on page 35.
- Verify that PeopleSoft security has been set correctly, and that the Component Interface is listed on the Permission List, and that time-out has been configured for the connector's user account. For more information, see "Configuring time-out for the user account" on page 13.
- Verify that Row-Level Security has been set for the connector's user account.
- Trace the API by selecting relevant options on the Trace tab in the PeopleSoft Configuration Manager.

Startup problems (WebSphere InterChange Server broker only)

When using the connector with WebSphere InterChange MQ Integrator Broker as the integration broker and you encounter difficulties starting the connector:

- Check to make sure that ICS is up and running.

  On UNIX, enter the following command:

  ```
  ICS_manager -stat
  ```

  On Windows, the second parameter after the `start_`*`connector`* command should contain the InterChange Server name. For example:

  `start_PeopleSoft` *`ConnectorName InterChangeServerName`*

  where `InterChangeServerName` is the name of the ICS instance.
- Follow the suggestions in "Startup problems" on page 73.

Startup problems (WebSphere MQ Integrator broker only)

When using the connector with MQ Integrator broker as the integration broker and you encounter difficulties starting the connector:

- Check to make sure that WebSphere MQ Integrator broker is up and running.

  On UNIX, enter the following command:

  `ICS_manager -stat`

  On Windows, the `start_connector` command must contain the name of the connector and the broker, as well as the path and name of the connector's configuration file. For example:

  `start_PeopleSoft ConnectorName BrokerName -cConfigFileName`

  where `BrokerName` is the name of the WebSphere MQ Integrator broker broker and *`ConfigFileName`* specifies the path and filename of the connector's configuration file.
- Follow the suggestions in "Startup problems" on page 73.

Processing Problems

If an error occurs when the connector is processing, create a simple Java program and test the following using IDE:

1. The connection to the application
2. The `Get()`, `Create()`, and `Find()` methods
3. The retrieval of Property values from each level of the corresponding Component Interface

This test can isolate problems arising between the PeopleSoft API and the connector. In other words, if you have no problems in the test program, then there probably is a problem with the connector code.

PeopleSoft generates a template for a test Java program. To generate the template:

1. Open the Component Interface in question.
2. Right click anywhere in the window and select Generate Java Template from the popup menu.

Mapping (ICS Integration Broker Only)

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

Error Handling and Logging

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

Error Types

Table 16 describes the types of tracing messages that the connector outputs at each trace level. These messages are in addition to any tracing messages output by the WebSphere business integration system's architecture, such as the Java connector

execution wrapper and the IBM MQSeries message interface.

Table 16. PeopleSoft Tracing Messages

Tracing Level	Tracing Messages
Level 0	Message that identifies the connector version. No other tracing is done at this level. This is the default value.
Level 1	• Status messages • Messages that provide identifying (key) information for each business object processed • Messages delivered each time the pollForEvents method is executed
Level 2	• Business object handler messages that contain information such as the arrays and child business objects that the connector encounters or retrieves during the processing of a business object • Messages logged each time a business object is posted to the integration broker, either from `gotApplEvent()` or `consumeSync()` • Messages that indicate that a business object has been received as a request
Level 3	• Foreign key processing messages that contain such information as when the connector has found or has set a foreign key in a business object • Messages that provide information about business object processing. For example, these messages are delivered when the connector finds a match between business objects, or finds a business object in an array of child business objects
Level 4	• Application-specific information messages, for example, messages showing the values returned by the functions that parse the business object's application-specific information fields • Messages that identify when the connector enters or exits a function, which helps trace the process flow of the connector • All thread-specific messages. If the connector spawns multiple threads, a message appears for the creation of each new thread.
Level 5	• Messages that indicate connector initialization, for example, messages showing the value of each configuration property retrieved from the integration broker. • Messages that include statements executed in the application. At this trace level, the connector log file contains all statements executed in the destination application and the value of any variables that are substituted. • Messages that comprises a representation of a business object before the connector begins processing it (displaying its state as the connector receives it) and after the connector has completed its processing (displaying its state as the connector returns it to the integration broker) • Messages that comprise a business object dump • Messages that indicate the status of each thread the connector spawns while it is running

Error Messages

All the error messages that the connector generates are stored in a message file named `PeopleSoftConnector.txt`. Each error has an error number followed by the error message. For example:

```
1210
```

```
PeopleSoft Connector unable to initialize.
1211
PeopleSoft Connector failed to locate.
```

Loss of Connection to the Application

If the connector's application-specific component fails to establish connection, it sends `FAIL` to integration broker and terminates.

Chapter 6. Upgrading the connector

If you have installed an earlier version of the IBM WebSphere Business Integration Adapter for PeopleSoft, you must delete it before installing this connector.

Also, upgrade or delete all previous customizations made in PeopleSoft for the connector according to the project standards for PeopleSoft Upgrades and Fixes. Each version of the connector is included with an Application Designer project (a maintenance project) containing all current PeopleSoft objects and code needed for Event Notification. The project also lists adapter-specific fixes and enhancements made to those objects in the details portion of the project properties. Follow the general project standards for importing or copying this project into databases. It might be necessary to delete menus, pages, records or fields if they are no longer used.

To install the new connector, follow the instructions in Chapter 2, “Installing and configuring the connector,” on page 11.

Appendix A. Standard configuration properties for connectors

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

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

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

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

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

New and deleted properties

These standard properties have been added in this release.

New properties

- XMLNameSpaceFormat

Deleted properties

- RestartCount

Configuring standard connector properties

Adapter connectors have two types of configuration properties:

- Standard configuration properties
- Connector-specific configuration properties

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

Using Connector Configurator

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

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

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

Setting and updating property values

The default length of a property field is 255 characters.

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

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

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

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

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

Summary of standard properties

Table 17 on page 81 provides a quick reference to the standard connector configuration properties. Not all the connectors make use of all these properties, and property settings may differ from integration broker to integration broker, as standard property dependencies are based on `RepositoryDirectory`.

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

Table 17. Summary of standard configuration properties

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

Table 17. Summary of standard configuration properties (continued)

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

Table 17. Summary of standard configuration properties (continued)

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

Table 17. Summary of standard configuration properties (continued)

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

Standard configuration properties

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

AdminInQueue

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

The default value is CONNECTORNAME/ADMININQUEUE.

AdminOutQueue

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

The default value is CONNECTORNAME/ADMINOUTQUEUE.

AgentConnections

Applicable only if RepositoryDirectory is <REMOTE>.

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

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

AgentTraceLevel

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

ApplicationName

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

BrokerType

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

CharacterEncoding

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

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

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

ConcurrentEventTriggeredFlows

Applicable only if `RepositoryDirectory` is <REMOTE>.

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

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

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

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

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

ContainerManagedEvents

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

The default value is No value.

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

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

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

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

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

ControllerStoreAndForwardMode

Applicable only if RepositoryDirectory is <REMOTE>.

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

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

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

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

The default is true.

ControllerTraceLevel

Applicable only if RepositoryDirectory is <REMOTE>.

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

DeliveryQueue

Applicable only if DeliveryTransport is JMS.

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

The default value is CONNECTORNAME/DELIVERYQUEUE.

DeliveryTransport

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

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

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

WebSphere MQ and IDL

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

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

JMS

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

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

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

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

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

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

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

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

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

DuplicateEventElimination

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

This property can also be set to false.

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

FaultQueue

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

The default value is CONNECTORNAME/FAULTQUEUE.

JvmMaxHeapSize

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

The default value is 128m.

JvmMaxNativeStackSize

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

The default value is 128k.

JvmMinHeapSize

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

The default value is 1m.

jms.FactoryClassName

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

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

jms.MessageBrokerName

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

The default is `crossworlds.queue.manager`.

jms.NumConcurrentRequests

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

The default value is 10.

jms.Password

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

There is no default.

jms.UserName

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

There is no default.

ListenerConcurrency

This property supports multi-threading in MQ Listener when ICS is the integration broker. It enables batch writing of multiple events to the database, thus improving system performance. The default value is 1.

This property applies only to connectors using MQ transport. The `DeliveryTransport` property must be set to `MQ`.

Locale

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

A locale name has the following format:

ll_TT.codeset

where:

ll	a two-character language code (usually in lower case)

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 Connector Configurator.

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

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

LogAtInterchangeEnd

Applicable only if RespositoryDirectory is <REMOTE>.

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

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

MaxEventCapacity

The maximum number of events in the controller buffer. This property is used by flow control and is applicable only if the value of the `RepositoryDirectory` property is <REMOTE>.

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

MessageFileName

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

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

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

MonitorQueue

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

The default value is CONNECTORNAME/MONITORQUEUE

OADAutoRestartAgent

Valid only when the RepositoryDirectory is <REMOTE>.

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

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

The default value is false.

OADMaxNumRetry

Valid only when the RepositoryDirectory is <REMOTE>.

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

The default value is 1000.

OADRetryTimeInterval

Valid only when the RepositoryDirectory is <REMOTE>.

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

The default is 10.

PollEndTime

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

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

PollFrequency

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

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

The default is 10000.

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

PollQuantity

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

PollStartTime

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

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

RequestQueue

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

The default value is CONNECTOR/REQUESTQUEUE.

RepositoryDirectory

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

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

When the integration broker is a WebSphere message broker or WAS, this value must be set to <*local directory*>.

ResponseQueue

Applicable only if DeliveryTransport is JMS and required only if RepositoryDirectory is <REMOTE>.

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

RestartRetryCount

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

The default is 3.

RestartRetryInterval

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

The default is 1.

RHF2MessageDomain

WebSphere message brokers and WAS only.

This property allows you to configure the value of the field domain name in the JMS header. When data is sent to WMQI over JMS transport, the adapter framework writes JMS header information, with a domain name and a fixed value of `mrm`. A connfigurable domain name enables users to track how the WMQI broker processes the message data.

A sample header would look like this:

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

The default value is `mrm`, but it may also be set to `xml`. This property only appears when DeliveryTransport is set to `JMS`and WireFormat is set to `CwXML`.

SourceQueue

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

Designates the JMS source queue for the connector framework in support of guaranteed event delivery for JMS-enabled connectors that use a JMS event store. For further information, see “ContainerManagedEvents” on page 86.

The default value is `CONNECTOR/SOURCEQUEUE`.

SynchronousRequestQueue

Applicable only if `DeliveryTransport` is JMS.

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

The default is `CONNECTORNAME/SYNCHRONOUSREQUESTQUEUE`

SynchronousResponseQueue

Applicable only if `DeliveryTransport` is JMS.

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

The default is CONNECTORNAME/SYNCHRONOUSRESPONSEQUEUE

SynchronousRequestTimeout

Applicable only if DeliveryTransport is JMS.

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

The default value is 0.

WireFormat

Message format on the transport.

- If the RepositoryDirectory is a local directory, the setting is CwXML.
- If the value of RepositoryDirectory is <REMOTE>, the setting isCwBO.

WsifSynchronousRequest Timeout

WAS integration broker only.

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

The default value is 0.

XMLNameSpaceFormat

WebSphere message brokers and WAS integration broker only.

A strong property that allows the user to specify short and long name spaces in the XML format of business object definitions.

The default value is short.

Appendix B. Connector Configurator

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

You use Connector Configurator to:

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

Note:

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

The topics covered in this appendix are:

Overview of Connector Configurator

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

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

You use Connector Configurator to:

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

The mode in which you run Connector Configurator, and the configuration file type you use, may differ according to which integration broker you are running. For example, if WMQI is your broker, you run Connector Configurator directly, and not from within System Manager (see "Running Configurator in stand-alone mode" on page 96).

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

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

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

For **connector-specific properties**, however, you need first to define the properties and then set their values. You do this by creating a connector-specific property template for your particular adapter. There may already be a template set up in your system, in which case, you simply use that. If not, follow the steps in "Creating a new template" on page 97 to set up a new one.

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

Starting Connector Configurator

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

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

Running Configurator in stand-alone mode

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

To do so:

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

You may choose to run Connector Configurator independently to generate the file, and then connect to System Manager to save it in a System Manager project (see "Completing a configuration file" on page 101.)

Running Configurator from System Manager

You can run Connector Configurator from System Manager.

To run Connector Configurator:

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

To edit an existing configuration file:

1. In the System Manager window, select any of the configuration files listed in the Connector folder and right-click on it. Connector Configurator opens and displays the configuration file with the integration broker type and file name at the top.
2. Click the Standard Properties tab to see which properties are included in this configuration file.

Creating a connector-specific property template

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

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

- To create a new template, see "Creating a new template" on page 97.
- 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 **Dependences - Connector-Specific Property Template** dialog box appears.

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

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

   == (equal to)

   != (not equal to)

   > (greater than)

   < (less than)

   >= (greater than or equal to)

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

Creating a new configuration file

When you create a new configuration file, your first step is to select an integration broker. The broker you select determines the properties that will appear in the configuration file.

To select a broker:

- In the Connector Configurator home menu, click **File>New>Connector Configuration**. The **New Connector** dialog box appears.
- In the I**ntegration Broker** field, select ICS, WebSphere Message Brokers or WAS connectivity.
- Complete the remaining fields in the **New Connector** window, as described later in this chapter.

You can also do this:

- In the System Manager window, right-click on the **Connectors** folder and select **Create New Connector**. Connector Configurator opens and displays the **New Connector** dialog box.

Creating a configuration file from a connector-specific template

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

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

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

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

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

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

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

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

Using an existing file

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

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

- A previous configuration file for the connector.
  Such a file typically has the extension *.cfg.

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

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

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

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

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

     Choose this option if a `*.txt` file was delivered in the adapter package for the connector, or if a definition file is available under another extension.
3. In the directory display, navigate to the appropriate connector definition file, select it, and click **Open**.

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

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

Completing a configuration file

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

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

1. Under the **Standard Properties** tab, select the value field for the `BrokerType` property. In the drop-down menu, select the value `ICS`, `WMQI`, or `WAS`.
2. The Standard Properties tab will display the properties associated with the selected broker. You can save the file now or complete the remaining configuration fields, as described in “Specifying supported business object definitions” on page 104..
3. When you have finished your configuration, click **File>Save>To Project** or **File>Save>To File**.

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

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

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

Setting the configuration file properties

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

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

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

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

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

- Associated Maps
- Resources
- Messaging (where applicable)

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

Standard properties differ from connector-specific properties as follows:

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

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

- A field with a grey background indicates a standard property. You can change the value but cannot change the name or remove the property.
- A field with a white background indicates an application-specific property. These properties vary according to the specific needs of the application or connector. You can change the value and delete these properties.
- Value fields are configurable.
- The **Update Method** field is informational and not configurable. This field specifies the action required to activate a property whose value has changed.

Setting standard connector properties

To change the value of a standard property:

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

Setting application-specific configuration properties

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

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

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

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

Encryption for connector properties

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

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

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

Update method

Refer to the descriptions of update methods found in the *Standard configuration properties for connectors* appendix, under "Setting and updating property values" on page 80.

Specifying supported business object definitions

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

Note: Some connectors require that certain business objects be specified as supported in order to perform event notification or additional configuration (using meta-objects) with their applications. For more information, see the *Connector Development Guide for C++* or the *Connector Development Guide for Java*.

If ICS is your broker

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

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

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

To delete a business object from the supported list:

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

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

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

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

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

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

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

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

If a WebSphere Message Broker is your broker

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

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

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

If WAS is your broker

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

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

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

Associated maps (ICS only)

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

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

will be used to transform the application-specific business object to the generic business object or the generic business object to the application-specific business object.

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

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

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

- **Business Object Name**

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

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

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

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

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

  To explicitly bind a map:

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

Resources (ICS)

The **Resource** tab allows you to set a value that determines whether and to what extent the connector agent will handle multiple processes concurrently, using connector agent parallelism.
Not all connectors support this feature. If you are running a connector agent that was designed in Java to be multi-threaded, you are advised not to use this feature, since it is usually more efficient to use multiple threads than multiple processes.

Messaging (ICS)

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

Setting trace/log file values

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

To change the logging and tracing values:

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

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

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

Data handlers

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

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

Saving your configuration file

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

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

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

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

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

Changing a configuration file

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

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

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

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

Completing the configuration

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

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

Using Connector Configurator in a globalized environment

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

Connector Configurator supports non-English characters in:

- All value fields
- Log file and trace file path (specified in the **Trace/Log files** tab)

The drop list for the `CharacterEncoding` and `Locale` standard configuration properties displays only a subset of supported values. To add other values to the drop list, you must manually modify the `\Data\Std\stdConnProps.xml` file in the product directory.

For example, to add the locale `en_GB` to the list of values for the `Locale` property, open the `stdConnProps.xml` file and add the line in boldface type below:

```
<Property name="Locale"
isRequired="true"
updateMethod="component restart">
                <ValidType>String</ValidType>
            <ValidValues>
                                <Value>ja_JP</Value>
                                <Value>ko_KR</Value>
                                <Value>zh_CN</Value>
                                <Value>zh_TW</Value>
                                <Value>fr_FR</Value>
                                <Value>de_DE</Value>
                                <Value>it_IT</Value>
                                <Value>es_ES</Value>
                                <Value>pt_BR</Value>
                                <Value>en_US</Value>
                                <Value>en_GB</Value>
                    <DefaultValue>en_US</DefaultValue>
            </ValidValues>
    </Property>
```

Appendix C. Notices

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

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

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

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

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

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

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

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

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

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

Burlingame, CA 94010
U.S.A

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

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

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

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

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

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

Programming interface information

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

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

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

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

Trademarks and service marks

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

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

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

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

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

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

IBM WebSphere Business Integration Adapter Framework V2.4.0