IBM WebSphere Business Integration Adapters

IBM

Adapter for PeopleSoft User Guide

Adapter Version 3.1.0

IBM WebSphere Business Integration Adapters

Adapter for PeopleSoft User Guide

Adapter Version 3.1.0

30September2004

This edition of this document applies to WebSphere Business Integration Adapter for PeopleSoft (5724-G98), version 3.1.0.

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

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

Contents

About this document

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

What this document includes

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

What this document does not include

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

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

Audience

This document is for 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.

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

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`.

New in this release

New in Release 3.1.0

The following are new in this release:

- The adapter supports the following child verbs: insert, update, and delete.
- Support for PeopleTools 8.44 and 8.45 is added.
- Delta event propagation is provided for PeopleTools, versions 8.44 and 8.45.
- The adapter for PeopleSoft is supported on the Solaris 9.0 and the Windows 2003 platforms. In addition, the adapter is enabled to run on Linux Red Hat 3.0. After PeopleSoft client API tools are enabled for Linux Red Hat 3.0, look for updates about using the adapter on this platform.
- The adapter now supports bidirectional script languages.
- Support for result set retrieval, which is enabled only with specific versions of DB2® Information Integrator broker (For information about these specific versions, refer to the DB2 Information Integrator product documentation.)

New in Release 3.0.x

The following are new in this release:

- The Adapter for PeopleSoft provides additional support for processing attributes by the order in which they are listed in the business object definition.
- Polling for future effective-dated events is added.
- As of version 3.0.x, the adapter for PeopleSoft is not supported on Solaris 7, so references to that platform version have been deleted from this guide.

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 8.14 through 8.43 are supported. However, features that are specific to certain versions of PeopleTools are only supported against those versions.
- 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 2.3 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. (PingCompInterface)
- This connector adds a new effective-dating sequence attribute that allows data to be inserted based on the sequence number.

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 versions 8.18 through 8.40 of PeopleTools.
- 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 6, "Generating business object definitions using PeopleSoftODA," on page 63.

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 7, "Importing the project" on page 14, "Code for processing application events" on page 35, "Event and archive tables" on page 37.
- Four new connector-specific configuration properties: "ConvertToPrimitiveFloat" on page 116, "EventKeyDelimiter" on page 116, "SetLangCode" on page 117, and "ReconnectSessionOnGetFail" on page 117.

Chapter 1. Overview

This chapter defines useful adapter terminology, and it 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.x applications that use PeopleTools, versions 8.45.

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 chapter contains the following sections:

- "Terminology"
- "Connector components" on page 2
- "How the connector works" on page 4

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*.

Terminology

The following terms are used in this document.

adapter
> The component in the WebSphere business integration system that provides components to support communication between an integration broker and either an application or a technology. An adapter always includes a connector, message files, and configuration tools. It can also include an Object Discovery Agent (ODA) or a data handler.

adapter framework
> The software that IBM provides to configure and run an adapter. The runtime components of the adapter framework include the Java runtime environment, the connector framework, and the Object Discovery Agent (ODA) runtime. This connector framework includes the connector libraries (C++ and Java™) needed to develop new connectors. The ODA runtime includes the library in the Object Development Kit (ODK) needed to develop new ODAs. The configuration components include the following tools:
> - Business Object Designer,
> - Connector Configurator,
> - Log Viewer,
> - System Manager,
> - Adapter Monitor,
> - Test Connector

- and, optionally, any Object Discovery Agents (ODAs) associated with an adapter.

Adapter Development Kit (ADK)
A development kit that provides some samples for adapter development, including sample connectors and Object Discovery Agents (ODAs).

connector
The component of an adapter that uses business objects to send information about an event to an integration broker (event notification) or receive information about a request from the integration broker (request processing). A connector consists of the connector framework and the connector's application-specific component.

connector framework
The component of a connector that manages interactions between a connector's application-specific component and the integration broker. This component provides all required management services and retrieves the meta-data that the connector requires from the repository. The connector framework, whose code is common to all connectors, is written in Java and includes a C++ extension to support application-specific components written in C++.

connector controller
The subcomponent of the connector framework that interacts with collaborations. A connector controller runs within InterChange Server and initiates mapping between application-specific and generic business objects, and manages collaboration subscriptions to business object definitions.

integration broker
The component in the WebSphere business integration system that integrates data among heterogeneous applications. An integration broker typically provides a variety of services that include: the ability to route data, a repository of rules that govern the integration process, connectivity to a variety of applications, and administrative capabilities that facilitate integration. Examples of integration brokers: the WebSphere Business Integration Message Broker; WebSphere Business InterChange Server.

WebSphere business integration system
An enterprise solution that moves information among diverse sources to perform business exchanges, and that processes and routes information among disparate applications in the enterprise environment. The business integration system consists of an integration broker and one or more adapters.

Connector components

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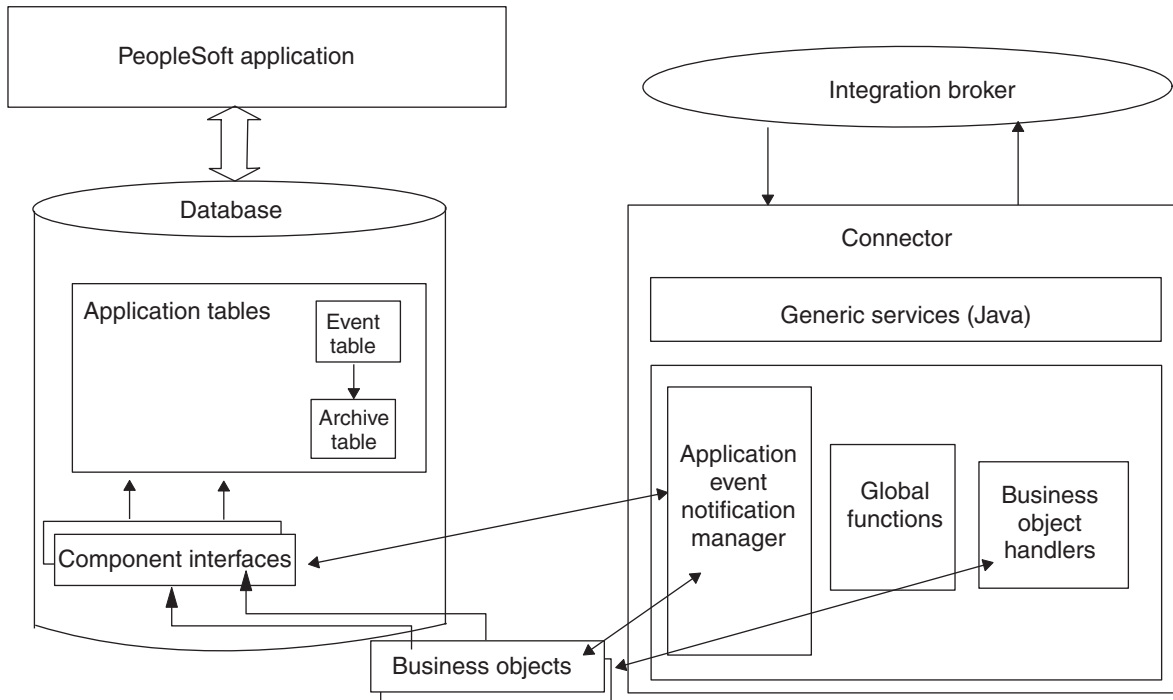
Figure 1. Connector architecture

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.

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. For information on creating a Component Interface and its corresponding business object, see "Component interface and business object relationship" on page 40. For information on creating the required classes and methods, see "Generating APIs" on page 45.

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 7, "cw_publish_event() and cw_publish_future_dated_events() Functions" on page 35, and "Event and archive tables" on page 37.

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

How the connector works

This section describes:
- "Interacting with the PeopleSoft application"
- "Processing business object requests" on page 4
- "Processing application events" on page 7

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 5, "Understanding business objects for the connector," on page 39

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 60.

     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 5, "Understanding business objects for the connector," on page 39. For information on connector-specific properties, see Appendix B, "Connector specific properties," on page 115.

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 6
- "Processing update requests" on page 6
- "Processing delete requests" on page 7

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 50.

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 51.

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 53.

Processing deltaupdate requests

When the integration broker sends a business object request with the deltaupdate 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 deltaupdate verb is supported only for top-level business objects. Associated child-level business objects should have one of these three verbs: insert, delete, or update. The child verb operations will be performed recursively throughout all levels of children in the hierachy, and the top-level object will be updated in the application. See "DeltaUpdate operations" on page 55 for more information.

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 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 8—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 eithe the `cw_publish_event()` or the `cw_publish_future_dated_event()`function, which are stored in the FieldFormula event of the FUNCLIB_CW record. You must declare and call one of these functions 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_event() and cw_publish_future_dated_events() Functions" on page 35.

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 INPROGRESS events are reset to READYFORPOLL (a value of 0) during the connector's initialization.

The connector uses the CWEVENT_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 37.

Event archiving

While the CWEVENT_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

The following steps describe the event notification process:

1. When a user saves a change online in a Component, the `SavePostChg()` method calls either the `cw_publish_event()` function or the `cw_publish_future_dated_events()` function.
2. The `cw_publish_event()` 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 `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.

Note: The cw_publish_future_dated_events() function performs similarly to the cw_publish_event()function, except that it uses an additional parameter. Use this function if you want to pass the any date attribute (usually the effective date) of the component being saved so that events with future dates are polled only when the date arrives. If the effective date is in the future, the event status is set to 99 and the event date is set to the future date. If not, the status is set to 0 and the date is set to the system date.

For more information, see "cw_publish_event() and cw_publish_future_dated_events() Functions" on page 35.

3. 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 CWEVENT_CI Component and invokes the Find() method.

Note: The connector uses a different thread from the one used for request processing.

4. 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.

5. 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.

6. After retrieving complete event data from the Component Interface, the connector instantiates the appropriate application-specific business object.

7. The connector sends the application-specific business object to the integration broker.

8. 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.

This adapter supports the processing of bidirectional script data for the Arabic and Hebrew languages when the adapter is run in a Windows environment. Bidirectional processing is not supported in non-Windows environments. To use the bidirectional capacity, you must configure the bidirectional standard properties. For more information, refer to the standard configuration properties for connectors in Appendix A.

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 91.

Common Event Infrastructure

This adapter is compatible with the IBMs Common Event Infrastructure, a standard for event management that permits interoperability with other IBM WebSphere event-producing applications. If Common Event Infrastructure support is enabled, events produced by the adapter can be received (or used) by another Common Event Infrastructure-compatible application.

For more information refer to the Common Event Infrastructure appendix in this guide.

Application Response Measurement

This adapter is compatible with the Application Response Measurement application programming interface (API), an API that allows applications to be managed for availability, service level agreements, and capacity planning. An ARM-instrumented application can participate in IBM Tivoli® Monitoring for Transaction Performance, allowing collection and review of data concerning transaction metrics.

For more information refer to the Application Response Measurement appendix in this guide.

Chapter 2. Installing 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:
- "Adapter for PeopleSoft environment"
- "Installing the adapter and related files" on page 13
- "Verifying an installation" on page 13
- "Enabling the application for the connector" on page 14

Adapter for PeopleSoft environment

Before installing, configuring, and using the adapter, you must understand its environmental requirements.

Broker compatibility

This adapter runs with the WebSphere Business Integration Adapter FrameworkV2.6 and requires one of the following:
- WebSphere InterChange ServerV4.2.2,V4.3
- WebSphere MQ IntegratorV2.1
- WebSphere MQ Integrator BrokerV2.1
- WebSphere Business Integration Message BrokerV5.0.1
- WebSphere Application Server EnterpriseV5.0.2, in conjunction with WebSphere Studio Application Developer Integration EditionV5.0.1
- WebSphere Business Integration Server FoundationV5.1.1
- DB2 Information IntegratorV8.2.3.

Adapter platforms

In addition to a broker, this adapter requires one of the following operating systems:
- All operating system environments require the Java compiler (IBM JDK 1.4.2for Windows 2000) for compiling custom adapters
- **AIX®:**
  AIX 5.1 with Maintenance Level 4
  AIX 5.2 with Maintenance Level 1. This adapter supports 32-bit JVM on a 64-bit platform.
- **Solaris:**
  Solaris 8 (2.8)with Solaris Patch Cluster dated Feb. 11, 2004 or later
  Solaris 9 (2.9) with Solaris Patch Cluster dated February 11, 2004 or later. This adapter supports 32-bit JVM on a 64-bit platform.
- **HP-UX:**
  HP-UX 11.i (11.11) with June 2003 GOLDBASE11i and June 2003 GOLDAPPS11i bundles
- **Linux[R]:**
  Red Hat[R] Enterprise Linux AS 3.0 with Update 1

Red Hat Enterprise Linux ES 3.0 with Update 1
Red Hat Enterprise Linux WS 3.0 with Update 1

> **Note:** Although the adapter is supported on Red Hat Enterprise Linux 3.0, this platform cannot be used in production environments running PeopleTools versions 8.14 to 8.45. Future releases of PeopleTools are expected to support this platform. After PeopleSoft releases a version of PeopleTools that supports Red Hat Enterprise Linux 3.0, please check for more information and any software updates that may be required to use the adapter on this platform.

> **Note:** The TMTP (Tivoli Monitoring for Transaction Performance) component of the WebSphere Business Integration Adapter FrameworkV2.6 is not supported on Linux Red Hat.

- **Windows:**
  Windows 2000 (Professional, Server, or Advanced Server) with Service Pack 4
  Windows XP with Service Pack 1A, for WebSphere Business Integration Adapter Framework (administrative tools only)
  Windows 2003 (Standard Edition or Enterprise Edition)

Adapter dependencies

The adapter has the following software prerequisites and other dependencies:

- The connector requires two layers of PeopleSoft API classes. Make sure that these API classes are available:

  1. `psjoa.jar`— The connector uses this API to connect to the Application Server through BEA Systems' Jolt port, and to send objects through the Application Server. This file is included with PeopleTools. Save it to the directory where you will install the connector.

     For more information on retrieving PeopleSoft's API files, see PeopleSoft's *Installation and Administration Guide*.

  2. 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.

- A user account must exist in PeopleSoft for the connector, and it must be configured so that it does not time out. Also, it must use PeopleSoft security, and have privileges to insert, update, retrieve, and delete data from pages, components, and component interfaces in the PeopleSoft system.

- If WebSphere InterChange Server 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 WebSphere InterChange Server, the WebSphere InterChange Server connector will not run unless you manually switch the IBM WebSphere system environment variable to point to the desired product path.

- Required Fields, Pages, Records, Components, and Component Interfaces must be defined in your PeopleSoft application.

Installing the adapter and related files

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

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

Verifying an installation

The Installer copies the standard files associated with the adapter into your system. The utility installs the connector into the ProductDir\connectors\PeopleSoft directory, and adds a shortcut for the connector to the Start menu.This section describes the file structures after the product has been installed on a UNIX or Windows system.

Verifying installed files on a UNIX system

To verify the installation on a UNIX system, compare the files in the directory where you installed the adapter to those listed.

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.
/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. - Contains the `CW_EVENT_V140` project folder and the `CW_EVENT_V150` project folder. These projects contain the event notification objects.
connectors/messages	Contains the `PeopleSoftConnector.txt` file.
/bin/APP/DATA	Contains the PeopleSoft Connector template. This is used to create the connector configuration file using the Connector Configurator.

Verifying Installed files on a Windows system

To verify the installation on a Windows system, compare the files in the directory where you installed the adapter to those listed.

Table 2describes 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.
\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. • Contains the CW_EVENT_V140 project folder and the CW_EVENT_V150 project folder. These projects contain the event notification objects.
\connectors\messages	Contains the PeopleSoftConnector.txt file.
\bin\APP\DATA	Contains the PeopleSoft Connector template. This is used to create the connector configuration file using Connector Configurator.

Enabling the application for the connector

Before you can use the connector with your PeopleSoft application, you must import the appropriate project included with the adapter. The project contains the components required for event processing, building the required tables, and building the API files.

Importing the project

1. Navigate to the %*ProductDir*S%\connectors\PeopleSoft\dependencies directory that was created when you installed the adapter.
2. Select one of the following projects:
   • CW_EVENT_V140, if you are using PeopleTools versions 8.14 up to (but not including) 8.40.
   • CW_EVENT_V150, if you are using PeopleTools versions 8.40 through 8.45.
3. Use Application Designer to import the project.

Chapter 3. Configuring the connector

This chapter provides information about Connector Configurator.

Overview of Connector Configurator

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

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

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

You use Connector Configurator to:

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

The mode in which you run Connector Configurator, and the configuration file type you use, may differ according to which integration broker you are running. For example, if WMQI is your broker, you run Connector Configurator directly, and not from within System Manager (see "Running Configurator in stand-alone mode" on page 16).

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 17 to set up a new one.

Running connectors on UNIX

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

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

Starting Connector Configurator

You can start and run Connector Configurator in either of two modes:
* Independently, in stand-alone mode
* From System Manager

Running Configurator in stand-alone mode

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

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

You may choose to run Connector Configurator independently to generate the file, and then connect to System Manager to save it in a System Manager project (see "Completing a configuration file" on page 22.)

Running Configurator from System Manager

You can run Connector Configurator from System Manager.

To run Connector Configurator:

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

To edit an existing configuration file:

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

Creating a connector-specific property template

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

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

- To create a new template, see "Creating a new template" on page 17.
- To use an existing file, simply modify an existing template and save it under the new name. You can find existing templates in your `\WebSphereAdapters\bin\Data\App` directory.

Creating a new template

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

To create a template in Connector Configurator:

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

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

3. You can use an existing template whose property definitions are similar to those required by your connector as a starting point for your template. If you do not see any template that displays the connector-specific properties used by your connector, you will need to create one.

   - If you are planning to modify an existing template, select the name of the template from the list in the **Template Name** table below **Select the Existing Template to Modify: Find Template.**
   - This table displays the names of all currently available templates. You can also search for a template.

Specifying general characteristics

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

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

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

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

Specifying values

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

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

To create a new property value:

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

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

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

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

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

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

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

Setting dependencies

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

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

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

   == (equal to)

   != (not equal to)

   > (greater than)

   < (less than)

   >= (greater than or equal to)

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

Setting pathnames

Some general rules for setting pathnames are:

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

• Queue names may not have leading or embedded spaces.

Creating a new configuration file

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

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

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

To set the operating system for extended validation of the configuration file:
• Pull down the **Target System:** droplist on the menu bar.
• Select the operating system you are running on.

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

You also need to select an integration broker. The broker you select determines the properties that will appear in the configuration file. To select a broker:
• In the **Integration Broker** field, select ICS, WebSphere Message Brokers or WAS connectivity.
• Complete the remaining fields in the **New Connector** window, as described later in this chapter.

Creating a configuration file from a connector-specific template

Once a connector-specific template has been created, you can use it to create a configuration file:
1. Set the operating system for extended validation of the configuration file using the **Target System:** droplist on the menu bar (see "Creating a new configuration file" above).
2. Click **File>New>Connector Configuration**.
3. The **New Connector** dialog box appears, with the following fields:
   • **Name**

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

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

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

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

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

4. A configuration screen appears for the connector that you are configuring. The title bar shows the integration broker and connector name. You can fill in all the field values to complete the definition now, or you can save the file and complete the fields later.

5. To save the file, click **File>Save>To File** or **File>Save>To Project**. To save to a project, System Manager must be running.
If you save as a file, the **Save File Connector** dialog box appears. Choose `*.cfg` as the file type, verify in the File Name field that the name is spelled correctly and has the correct case, navigate to the directory where you want to locate the file, and click **Save**. The status display in the message panel of Connector Configurator indicates that the configuration file was successfully created.

Important: The directory path and name that you establish here must match the connector configuration file path and name that you supply in the startup file for the connector.

6. To complete the connector definition, enter values in the fields for each of the tabs of the Connector Configurator window, as described later in this chapter.

Using an existing file

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

- A connector definition file.
  This is a text file that lists properties and applicable default values for a specific connector. Some connectors include such a file in a `\repository` directory in their delivery package (the file typically has the extension `.txt`; for example, CN_XML.txt for the XML connector).

- An ICS repository file.
  Definitions used in a previous ICS implementation of the connector may be available to you in a repository file that was used in the configuration of that connector. Such a file typically has the extension `.in` or `.out`.

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

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

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

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

1. In Connector Configurator, click **File>Open>From File**.

2. In the **Open File Connector** dialog box, select one of the following file types to see the available files:

   - Configuration (`*.cfg`)
   - ICS Repository (`*.in`, `*.out`)

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

- All files (*.*)

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

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

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

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

Completing a configuration file

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

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

1. Under the **Standard Properties** tab, select the value field for the `BrokerType` property. In the drop-down menu, select the value ICS, WMQI, or WAS.
2. The Standard Properties tab will display the connector properties associated with the selected broker. The table shows **Property name**, **Value**, **Type**, **Subtype** (if the Type is a string), **Description**, and **Update Method**.
3. You can save the file now or complete the remaining configuration fields, as described in "Specifying supported business object definitions" on page 25..
4. When you have finished your configuration, click **File>Save>To Project** or **File>Save>To File**.

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

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

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

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

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

Setting the configuration file properties

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

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

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

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

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

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

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

Standard properties differ from connector-specific properties as follows:

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

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

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

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

Setting standard connector properties

To change the value of a standard property:

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

   Note: If the property has a Type of String, it may have a subtype value in the Subtype column. This subtype is used for extended validation of the property.

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

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

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

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

Setting connector-specific configuration properties

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

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

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

3. To encrypt a property, select the **Encrypt** box.

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

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

5. Choose to save or discard changes, as described for "Setting standard connector properties" on page 24.

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

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

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

Encryption for connector properties

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

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

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

Update method

Refer to the descriptions of update methods found in the Standard Properties appendix.

Specifying supported business object definitions

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

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

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

If ICS is your broker

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

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

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

To delete a business object from the supported list:

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

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

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

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

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

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

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

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

If a WebSphere Message Broker is your broker

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

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

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

If WAS is your broker

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

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

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

Associated maps (ICS)

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

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

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

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

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

*   **Business Object Name**

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

*   **Associated Maps**

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

- **Explicit Binding**

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

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

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

To explicitly bind a map:

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

Resources (ICS)

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

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

Messaging (ICS)

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

Validating messaging queues

Before you can validate a messaging queue, you must:

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

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

Security (ICS)

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

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

The **Keystore Target System Absolute Pathname** is:
- For Windows:
  `<ProductDir>\connectors\security\<connectorname>.jks`
- For UNIX:
  `opt/IBM/WebSphereAdapters/connectors/security/<connectorname>.jks`

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

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

The **Message Privacy Level** may be set as follows for the three messages categories (All Messages, All Administrative Messages, and All Business Object Messages):
- "" is the default; used when no privacy levels for a message category have been set.
- none
  Not the same as the default: use this to deliberately set a privacy level of none for a message category.
- integrity
- privacy
- integrity_plus_privacy

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

Before you can import a certificate into the adapter keystore, you must export it from the server keystore. When you select **Export Adapter Public Key**, the Export Adapter Public Key dialog box appears.
- The export certificate defaults to the same value as the keystore, except that the file extension is <filename>.cer.

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

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

Setting trace/log file values

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

To change the logging and tracing values:

1. Click the **Trace/Log Files** tab.
2. For either logging or tracing, you can choose to write messages to one or both of the following:

   - To console (STDOUT):
     Writes logging or tracing messages to the STDOUT display.

     Note: You can only use the STDOUT option from the **Trace/Log Files** tab for connectors running on the Windows platform.

   - To File:
     Writes logging or tracing messages to a file that you specify. To specify the file, click the directory button (ellipsis), navigate to the preferred location, provide a file name, and click **Save**. Logging or tracing message are written to the file and location that you specify.

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

Data handlers

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

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

Saving your configuration file

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

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

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

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

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

Changing a configuration file

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

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

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

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

Completing the configuration

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

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

Using Connector Configurator in a globalized environment

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

Connector Configurator supports non-English characters in:

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

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

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

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

Chapter 4. Configuring the application

This chapter describes the tasks you must complete to use your PeopleSoft[R] applications with the adapter. It includes the folowing topics:

- "Building required objects"
- "Building the API files for event processing"
- "Code for processing application events" on page 35
- "Sample PeopleCode declarations and function calls" on page 36
- "Event and archive tables" on page 37

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 2.
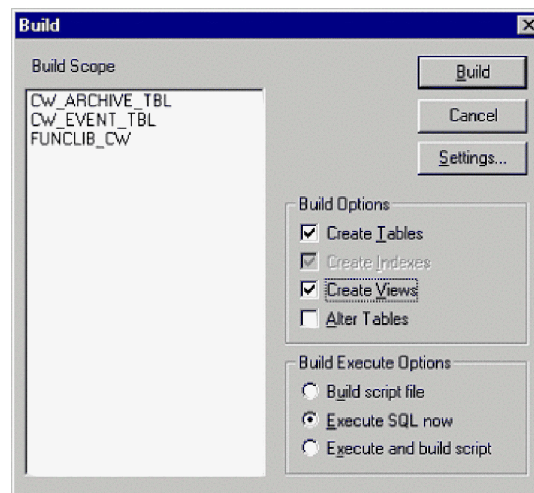
Figure 2. Building the tables

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 for event processing

To build the CWEVENT_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 CWEVENT_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 3 illustrates the screen within PeopleSoft.
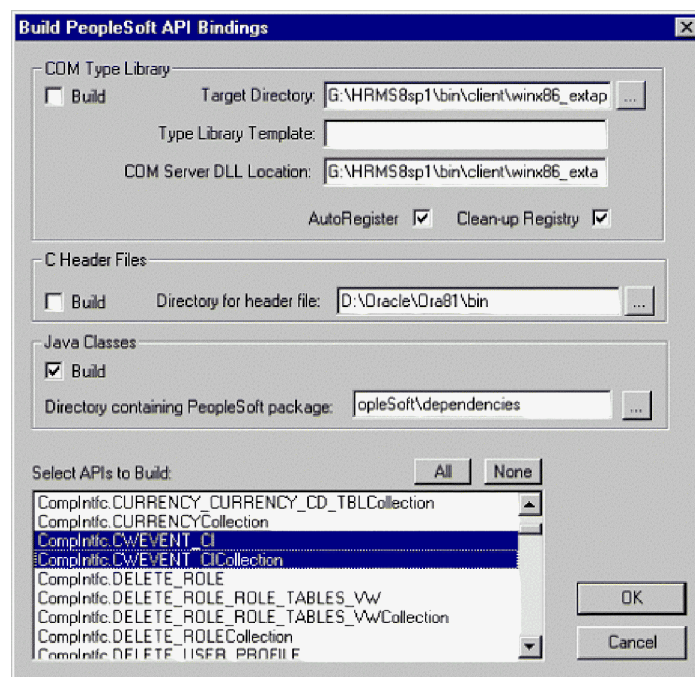
Figure 3. Building APIs

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 declarations and function calls" on page 36.

   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 declarations and function calls" on page 36.

Code for processing application events

The adapter includes the cw_publish_event() function and the cw_publish_future_dated_events()function in the FUNCLIB_CW function library. These functions perform similarly to insert events into the connector's event table. The difference between them is that cw_publish_future_dated_events()enables publishing events with a future effective-date to the event table, and cw_publish_event() does not. If you use events with a future-effective date, use the cw_publish_future_dated_events()function.

This section describes the functions and provides an example of the code that calls them.

cw_publish_event() and cw_publish_future_dated_events() Functions

The cw_publish_event() function takes four parameters, and the cw_publish_future_dated_events() function takes five parameters:

- &BONAME—contains the name of the WebSphere application-specific business object for PeopleSoft that is generated for the event. (Used by both functions.)
- &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 116.(Used by both functions.)
- &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. (Used by both functions.)
- &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". (Used by both functions.)

- &EFFDATE—the date associated with the event. If this value is greater than the system date, peoplecode in the cw_publish_future_dated_events function sets the event date to the &EFFDATE value . (This parameter is used by the cw_publish_future_dated_events.)

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 declarations and function calls

This section provides examples of the PeopleCode declarations and function calls for each of the two functions. Based on the function you use, insert one of the following code examples 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.

Code example for cw_publish_event:

```
/* 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;
```

Code example for cw_publish_future_dated events:

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

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

   &BONAME1 ="psft_CW_TEST_CI";
   &KEYLIST1 = "CW_PARENT_TBL.CW_PARENT_KEY1";
   &CWPRIORITY = 2;
   &CONNID1 ="PeopleSoftConnector";
   &EFFDATE = CW_PARENT_TBL.CW_PARENT_DTTM";

/* 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_future_dated_events(&BONAME1,&KEYLIST1,&CWPRIORITY1,&CONNID1,
&EFFDATE);

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	

Table 3. Event and archive table schema (continued)

Name	Description	Type	Constraint
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 116.	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	Not used.		
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. 99 (Future effective-dated events) This status is used only in the event table and not in the archive table.	VARCHAR	Not null

Chapter 5. Understanding business objects for the connector

This chapter contains the following sections:

- "Business object and attribute naming conventions"
- "Business object structure" on page 40
- "Creating a business object" on page 44
- "Business object verb processing" on page 47
- "Business object attribute properties" on page 56
- "Business object application-specific information" on page 58

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.

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, you may find that the PeopleSoft Collection has 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.

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 45.

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

Example Component and Component Interface

Figure 4 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.
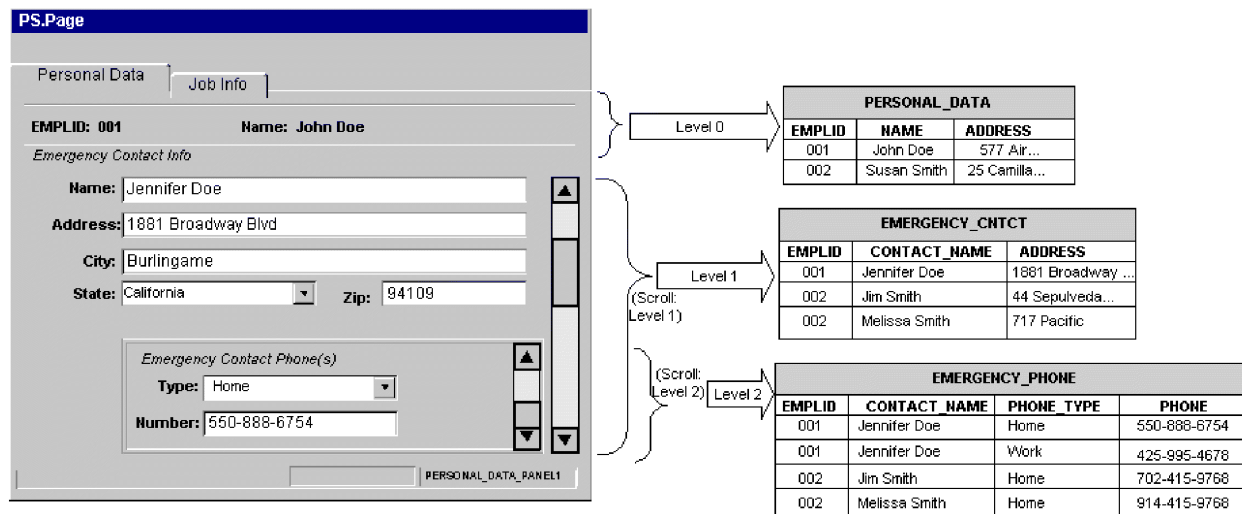
Figure 4. Example component

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 60.

Figure 5 illustrates the example Component Interface (`EMER_CONTACT_PROFILE`) that contains the `EMER_CONTACT` Component.
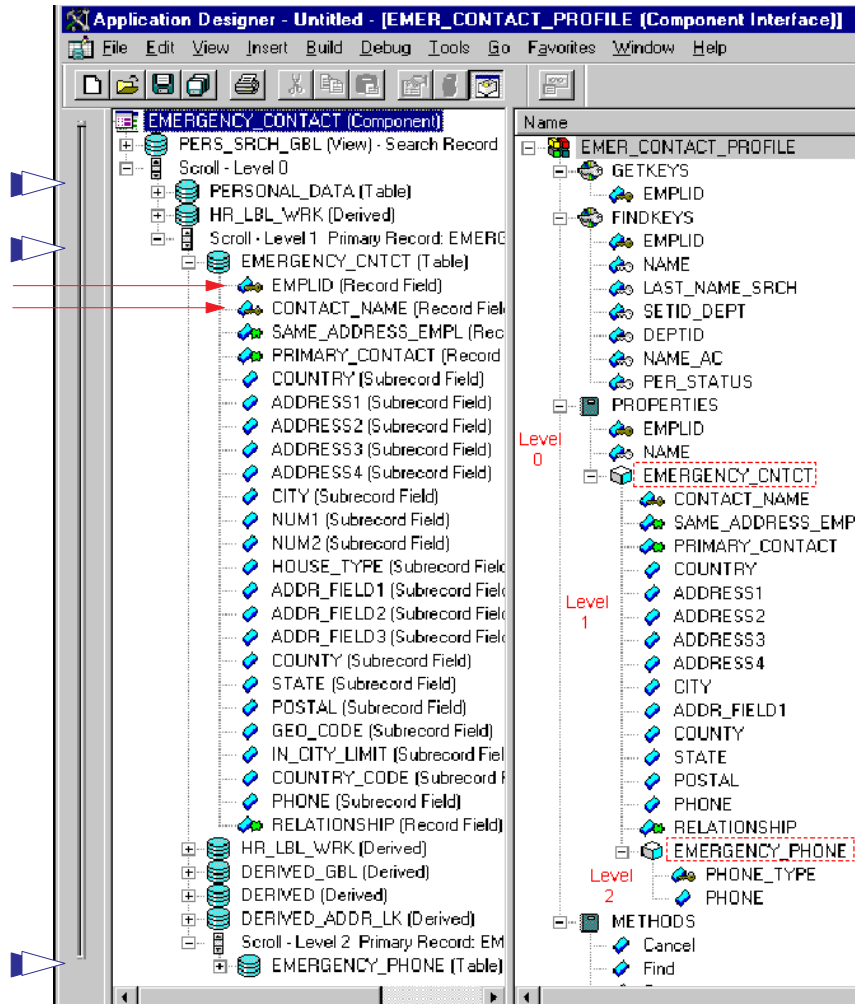
Figure 5. Example component interface

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

```
┌─────────────────────────┐          ┌─────────────────────────┐
│ PSFT_Emer gencyContact  │          │ PSFT_EmergencyPhone     │
├─────────────────────────┤          ├─────────────────────────┤
│ verb                    │─────────▶│ verb                    │
├─────────────────────────┤          ├─────────────────────────┤
│ EmpId                   │          │ EmpId                   │
│ ContactName             │          │ ContactName             │
│ DeptId                  │          │ PhoneType               │
│ Emer gencyPhone[n]      │          │ ...                     │
│ ...                     │          └─────────────────────────┘
└─────────────────────────┘
```
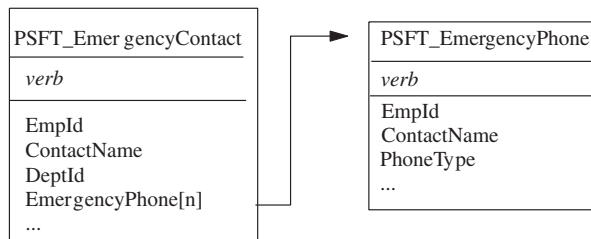
Figure 6. Example business objects

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 56 and "Business object application-specific information" on page 58. For more information about PeopleSoftODA, see Chapter 6, "Generating business object definitions using PeopleSoftODA," on page 63.
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

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 for event processing" on page 33.

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 58.

- 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 46
- "setFieldName() Method" on page 46
- "getCollectionName() Method" on page 47
- "CurrentItem() Method" on page 47
- "Item(index) Method" on page 47

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 5, 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 47.

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 58.

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:

- "After-images and deltas" on page 47, which defines the terms and explains how the connector works with after-images.
- "Verb Processing for business object requests" on page 49, which explains the steps the connector takes when creating, retrieving, updating, or deleting a business object.
- "Committing data" on page 56, which briefly explains how the connector saves data.

After-images and deltas

An after-image 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 after-images, 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 7 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.
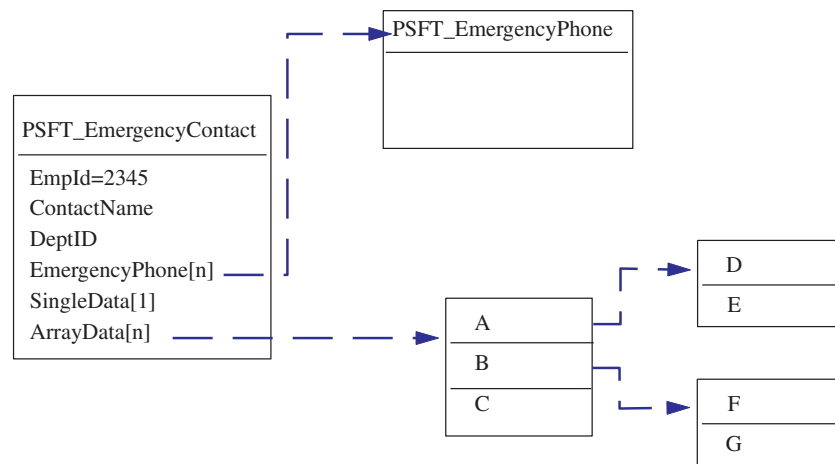
Figure 7. State of data prior to update

Figure 8 illustrates a business object request. This business object contains a new single-cardinality child business object and contains different business objects in its arrays.
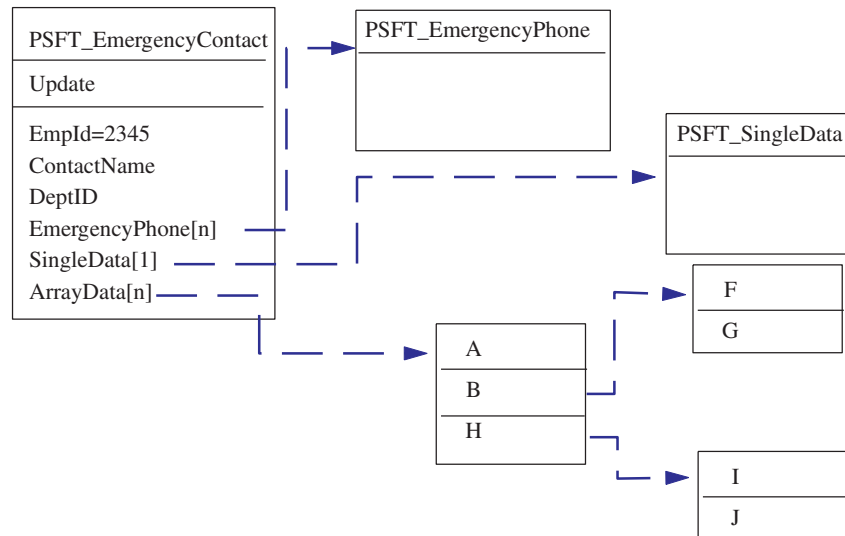
Figure 8. Data represented by an update request

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 60.

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 "PollFutureEvents" on page 116.

     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 57.

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 WebSphere InterChange Server is the integration broker, the ID must be delivered synchronously because WebSphere InterChange Server 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 56. For more information on specifying application-specific information, see "Application-specific information at the attribute level" on page 60.

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 60.) 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 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 60.

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.

DeltaUpdate operations

The deltaupdate verb is only supported for top-level business objects. When deltaupdate is set for the top-level business object, child objects of this parent should have one of the following three verbs set: insert, delete, or update. The verb operations for child objects will be performed recursively through all levels of children in the hierarchy.

The connector processes a deltaupdate verb request as follows:

- **Insert verb set for child objects:** The insert operation tries to insert the child that has been received. The operation is performed recursively, regardless of any verbs set for the child object's children. 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 processes the insert verb under deltaupdate operations very similarly to the way it processes a request to create a business object. For more information, see "Create operations" on page 50.

- **Delete verb set for child objects:** The delete operation tries to retrieve and delete the identified child. The operation is performed recursively, regardless of any verbs set for the child object's children. The connector processes the delete verb under deltaupdate operations very similarly to the way it processes a request to delete a business object. For more information, see "Delete operations" on page 56 and "Update operations" on page 53.

- **Update verb set for child objects:** The verb operation will try to retrieve and update the identified child. The connector processes the update verb under deltaupdate operations very similarly to the way it processes a request to update a business object. For more information, see "Update operations" on page 53.

Delete operations

The connector does not delete a top-level business object. However, it does physically delete a child business object under the following circumstances:

- The top-level business object uses the update verb, and the child does not exist in the request object that represents the source data, and `KeepRelationship` is set to false or not set at the child level.
- The top-level business object uses the deltaupdate verb. In this case, the delete operation will try to retrieve and delete the identified child.

For more information, see "Update operations" on page 53.

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 50 and "Update operations" on page 53.

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 60.

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.

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 50 and "Update operations" on page 53.

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

Table 4. 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]
:processAttributesInOrder=[true|false]
```

Table 5 describes these parameters.

Table 5. 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.
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. However, you should set this to true even for lower versions of PeopleTools. When using versions of PeopleTools lower than 8.4 that do not support this parameter, the adapter may log warnings. These warnings can be ignored. For information about this property, see your PeopleSoft documentation.

Table 5. AppSpecificInfo at the business-object level (continued)

AppSpecificinfo parameter	Description
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`.
processAttributesInOrder	Enables the adapter to process the attributes in the order in which they are listed in the business object definition.
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 6 describes each name-value parameter.

Table 6. 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 "PollFutureEvents" on page 116.
	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 WebSphere InterChange Server 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.
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.

Parameter	Description
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 6. 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"
- "Using PeopleSoftODA in business object designer" on page 66
- "Contents of the generated definition" on page 73
- "Sample business object definition file" on page 76
- "Modifying information in the business object definition" on page 81

Installation and usage

This section discusses the following:
- "Installing PeopleSoftODA"
- "Before using PeopleSoftODA" on page 63
- "Launching PeopleSoftODA" on page 64
- "Running multiple instances of PeopleSoftODA" on page 65
- "Working with error and trace message files" on page 65

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 WebSphere InterChange Server, 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 and configuring the application.

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 45.

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 7 are correct.

Table 7. 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 9 on page 67 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 67.

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 8 describes these values.

Table 8. 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 67.

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 9 on page 67 illustrates this window.

To select, configure, and run the ODA, follow these steps:
1. "Select the ODA" on page 67
2. "Configure initialization properties" on page 67

3. "Expand nodes and select component interfaces and collections" on page 69
4. "Confirm selection of objects" on page 70
5. "Generate the definition" on page 71 and, optionally, "Provide additional information" on page 71
6. "Save the definition" on page 72

Select the ODA

Figure 9 illustrates the first dialog box in Business Object Designer's six-step wizard. From this window, select the ODA to run.
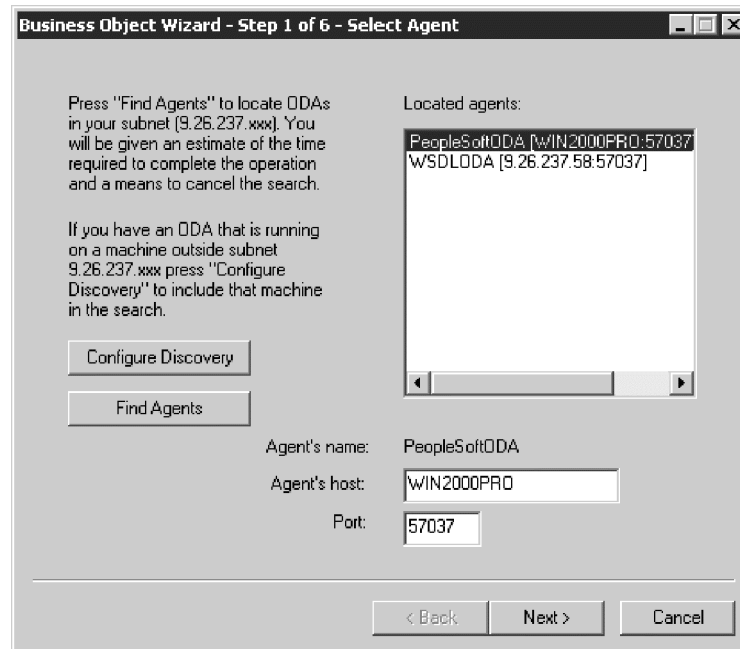
Figure 9. Selecting the ODA

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 10. 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*.
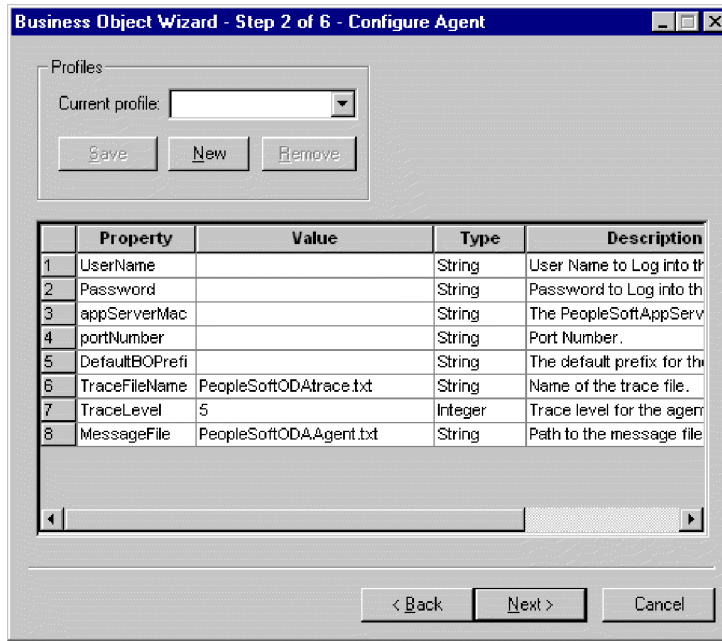
Figure 10. Configuring agent initialization properties

Configure the PeopleSoftODA properties described in Table 9.

Table 9. 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 71.
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.
7	TraceLevel	Integer	Level of tracing enabled for PeopleSoftODA. For more information, see "Working with error and trace message files" on page 65.

Table 9. PeopleSoftODA properties (continued)

Row number	Property name	Property type	Description
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 11 illustrates this dialog box with some Component Interfaces expanded.
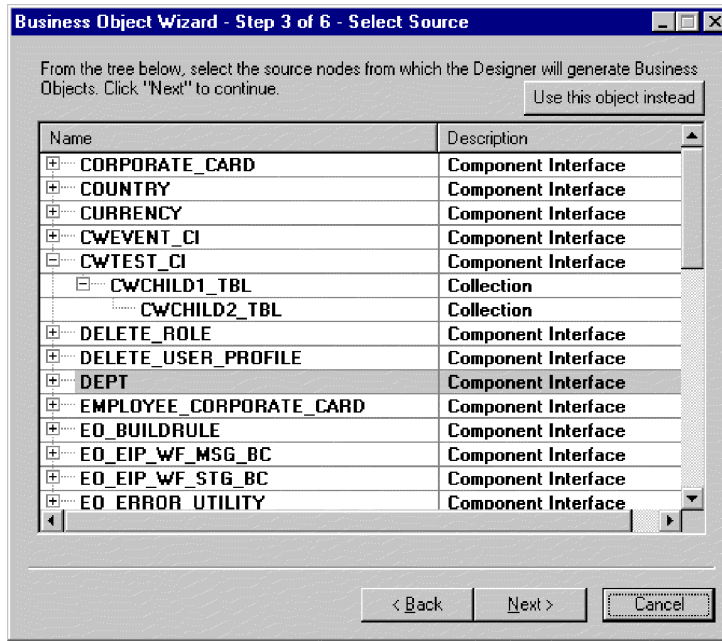
Figure 11. Tree of component interfaces with expanded nodes

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 12 illustrates this dialog box.
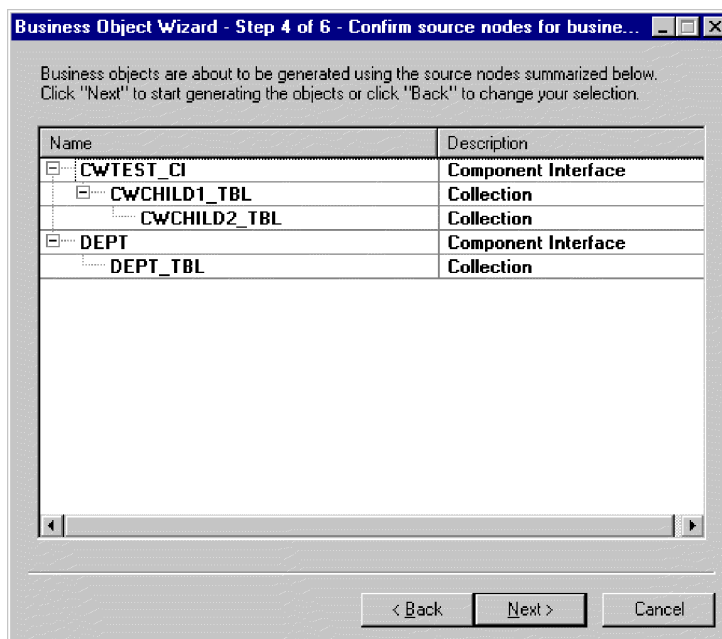
Figure 12. Confirming selection of objects

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 13 illustrates this dialog box.
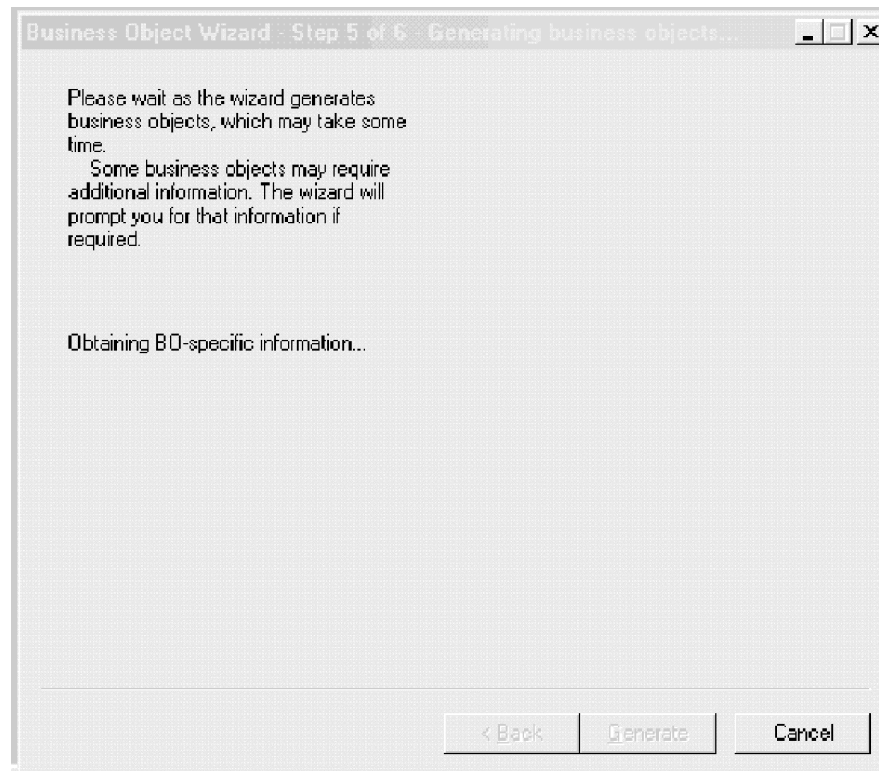
Figure 13. Generating the definition

Provide additional information

If the PeopleSoftODA needs additional information, Business Object Designer displays the BO Properties window, which prompts you for the information. Figure 14 illustrates this dialog box.
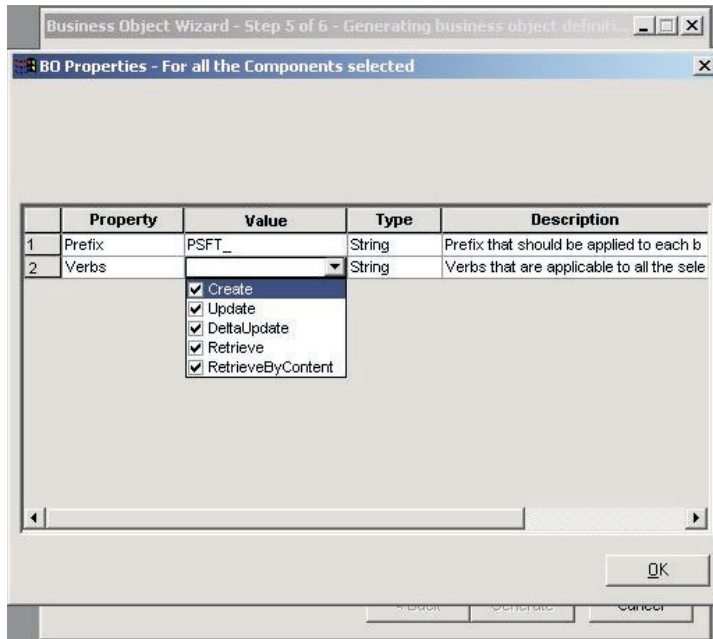
Figure 14. Providing additional Information

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 10 on page 68), 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 WebSphere InterChange Server 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 15 illustrates this dialog box.
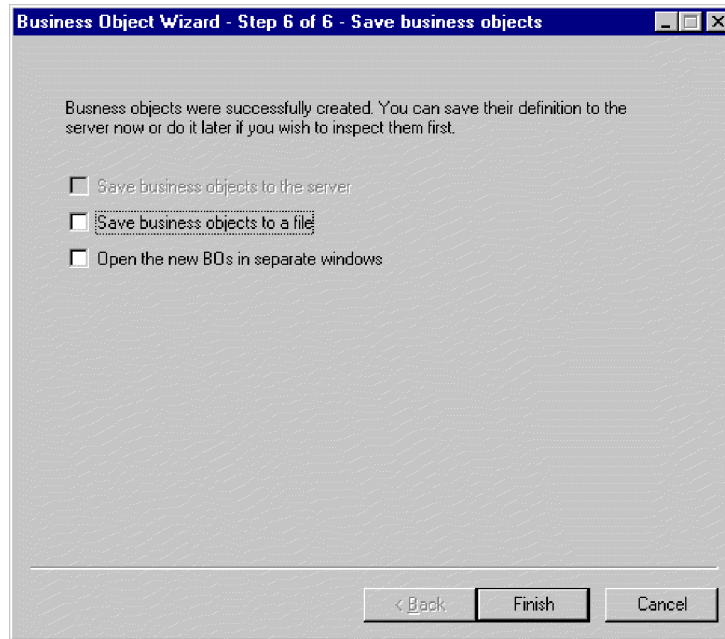
Figure 15. Saving the business object definition

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 14 on page 72)
- Application-specific information:
  - At the business-object level
  - For each attribute

This section describes:

- "Business-object-level properties"
- "Attribute properties" on page 74

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 10 describes these parameters.

Table 10. 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 58. 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 56.

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 11:

Table 11. 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 11, 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=getFieldName:set=setFieldName:UID=[true|false]:GetKey=[true|false]:KeepRelationship=[true|false]
```

PeopleSoftODA generates only those properties that are relevant to an attribute, as described in Table 12. If it generates more than one parameter, it separates parameters with a colon.

Table 12. 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=getPropertyName`	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=getCollectionName`	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=setPropertyName`	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 60. For more information on modifying definitions, see the *Business Object Development Guide*. For an example of using the parameters described in Table 12, see "Sample business object definition file" on page 76.

Verbs

Note: PeopleSoftODA generates the verbs specified in the BO Properties window (as illustrated in Figure 14 on page 72).

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 =
StartDate     Type = Date     MaxLength = 1     IsKey = false
IsForeignKey = false     IsRequired = true     AppSpecificInfo =
get=getStartDate:set=setStartDate     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     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     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 examples

The following example is the SavePostChange business object showing the function call `cw_publish_event`.

```
/* 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;
```

The following example is the SavePostChange business object showing the function call `cw_publish_future_dated_events`.

```
/* 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; &EFFDATE="DEPT_TBL_.STARTDATE"; 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_future_dated_events(&BONAME1, &KEYLIST1, &CWPRIORITY1, &CONNID,
&EFFDATEW);     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 5, "Understanding business objects for the connector," on page 39.

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 7. Running the connector

This chapter provides information about starting, stopping, and upgrading the connector.

- "Starting the connector"
- "Stopping the connector" on page 84
- "Upgrading the connector" on page 85

Starting the connector

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

ProductDir\connectors*connName*

where *connName* identifies the connector.

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

The name of the startup script depends on the operating-system platform, as Table 13 shows.

Table 13. Startup scripts for a connector

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

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

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

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

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

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

- From the command line

  – On Windows systems:

  start_*connName connName brokerName* [-c*configFile*]

  – On UNIX-based systems:

  connector_manager -start *connName brokerName* [-c*configFile*]

  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 (available only when the broker is WebSphere Application Server or InterChange Server), which is launched when you start System Manager

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

- From System Manager (available for all brokers)

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

- On Windows systems, you can configure the connector to start as a Windows service. In this case, the connector starts when the Windows system boots (for an Auto service) or when you start the service through the Windows Services window (for a Manual service).

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

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

Stopping the connector

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

- If you started the connector from the command line, with its connector startup script:
  - On Windows systems, invoking the startup script creates a separate "console" window for the connector. In this window, type "Q" and press Enter to stop the connector.
  - 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.

Upgrading the connector

IIf you have installed an earlier version of the 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 the connector," on page 11.

Chapter 8. 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"
- "Startup problems (WebSphere InterChange Server broker only)"
- "Startup problems (WebSphere MQ Integrator broker only)" on page 88
- "Processing Problems" on page 88
- "Mapping (WebSphere InterChange Server Only)" on page 88
- "Error Handling and Logging" on page 88
- "Loss of Connection to the Application" on page 90

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 45.
- Verify that PeopleSoft security has been set correctly; the Component Interface is listed on the Permission List; and that the connector's user account has been configured to never time out.
- 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 WebSphere InterChange Server 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 WebSphere InterChange Server name. For example:

  `start_PeopleSoft ConnectorName InterChangeServerName`

  where `InterChangeServerName` is the name of the WebSphere InterChange Server instance.
- Follow the suggestions in "Startup problems" on page 87.

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 87.

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 (WebSphere InterChange Server 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 14 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 14. 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.

Memory limitations with result set support

When result set support is being used on DB2, The adapter has a JVM memory restriction of 2GB for J2SE JRE 1.4.1, version SR2, that is provided with the adapter. To enable the result set process to utilize the 2GB memory, your environment must use hardware that facilitates efficient memory utilization without excessive paging.

PeopleSoft record limitations with result set retrieval

PeopleSoft applications are limited to returning the first 300 records per component when processing result set retrieval requests. Result set processing is supported only when DB2 Information Integrator is the integration broker.

Appendix A. Standard configuration properties for connectors

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

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

If your adapter supports DB2 Information Integrator, use the WMQI options and the DB2 II standard properties (see the Notes column in Table 15 on page 93.)

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

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

New properties

These standard properties have been added in this release:

- AdapterHelpName
- BiDi.Application
- BiDi.Broker
- BiDi.Metadata
- BiDi.Transformation
- CommonEventInfrastructure
- CommonEventInfrastructureContextURL
- ControllerEventSequencing
- jms.ListenerConcurrency
- jms.TransportOptimized
- ResultsSetEnabled
- ResultsSetSize
- TivoliTransactionMonitorPerformance

Standard connector properties overview

Connectors have two types of configuration properties:

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

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

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

Starting Connector Configurator

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

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

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

Configuration property values overview

The connector uses the following order to determine a property's value:
1. Default
2. Repository (valid only if WebSphere InterChange Server (ICS) is the integration broker)
3. Local configuration file
4. Command line

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

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

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

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

- **System restart**
  The new value takes effect only after you stop and restart the connector agent and the server.

To determine how a specific property is updated, refer to the **Update Method** column in the Connector Configurator window, or see the Update Method column in Table 15 on page 93.

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

- **ReposController**
  The property resides in the connector controller and is effective only there. If you change the value on the agent side, it does not affect the controller.

- **ReposAgent**
  The property resides in the agent and is effective only there. A local configuration can override this value, depending on the property.

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

Standard properties quick-reference

Table 15 provides a quick-reference to the standard connector configuration properties. Not all connectors require all of these properties, and property settings may differ from integration broker to integration broker.

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

Note: In the Notes column in Table 15, the phrase "RepositoryDirectory is set to <REMOTE>" indicates that the broker is InterChange Server. When the broker is WMQI or WAS, the repository directory is set to <*ProductDir*>\repository

Table 15. Summary of standard configuration properties

Property name	Possible values	Default value	Update method	Notes
AdapterHelpName	One of the valid subdirectories in <*ProductDir*>\bin\Data \App\Help\ that contains a valid <*RegionalSetting*> directory	Template name, if valid, or blank field	Component restart	Supported regional settings. Include `chs_chn`, `cht_twn`, `deu_deu`, `esn_esp`, `fra_fra`, `ita_ita`, `jpn_jpn`, `kor_kor`, `ptb_bra`, and `enu_usa` (default).
AdminInQueue	Valid JMS queue name	`<CONNECTORNAME>` `/ADMININQUEUE`	Component restart	This property is valid only when the value of DeliveryTransport is JMS
AdminOutQueue	Valid JMS queue name	`<CONNECTORNAME>` `/ADMINOUTQUEUE`	Component restart	This property is valid only when the value of DeliveryTransport is JMS

Table 15. Summary of standard configuration properties (continued)

Property name	Possible values	Default value	Update method	Notes
AgentConnections	1 through 4	1	Component restart	This property is valid only when the value of DeliveryTransport is MQ or IDL, the value of Repository Directory is set to <REMOTE> and the value of BrokerType is ICS.
AgentTraceLevel	0 through 5	0	Dynamic if broker is ICS; otherwise Component restart	
ApplicationName	Application name	The value specified for the connector application name	Component restart	
BiDi.Application	Any valid combination of these bidirectional attributes: 1st letter: I,V 2nd letter: L,R 3rd letter: Y, N 4th letter: S, N 5th letter: H, C, N	ILYNN (five letters)	Component restart	This property is valid only if the value of BiDi.Transforma tion is true
BiDi.Broker	Any valid combination of these bidirectional attributes: 1st letter: I,V 2nd letter: L,R 3rd letter: Y, N 4th letter: S, N 5th letter: H, C, N	ILYNN (five letters)	Component restart	This property is valid only if the value of BiDi.Transformation is true. If the value of BrokerType is ICS, the property is read-only.
BiDi.Metadata	Any valid combination of these bidirectional attributes: 1st letter: I,V 2nd letter: L,R 3rd letter: Y, N 4th letter: S, N 5th letter: H, C, N	ILYNN (five letters)	Component restart	This property is valid only if the value of BiDi.Transformation is true.
BiDi.Transformation	true or false	false	Component restart	This property is valid only if the value of BrokerType is not WAS .
BrokerType	ICS, WMQI, WAS	ICS	Component restart	
CharacterEncoding	Any supported code. The list shows this subset: ascii7, ascii8, SJIS, Cp949, GBK, Big5, Cp297, Cp273, Cp280, Cp284, Cp037, Cp437 .	ascii7	Component restart	This property is valid only for C++ connectors.

Table 15. Summary of standard configuration properties (continued)

Property name	Possible values	Default value	Update method	Notes
CommonEventInfrastruc ture	true or false	false	Component restart	
CommonEventInfrastruc tureURL	A URL string, for example, corbaloc:iiop: host:2809.	No default value.	Component restart	This property is valid only if the value of CommonEvent Infrastructure is true.
ConcurrentEventTrig geredFlows	1 through 32,767	1	Component restart	This property is valid only if the value of RepositoryDirectory is set to <REMOTE> and the value of BrokerType is ICS.
ContainerManagedEvents	Blank or JMS	Blank	Component restart	This property is valid only when the value of Delivery Transport is JMS.
ControllerEventSequenc ing	true or false	true	Dynamic	This property is valid only if the value of Repository Directory is set to <REMOTE> and the value of BrokerType is ICS.
ControllerStoreAndFor wardMode	true or false	true	Dynamic	This property is valid only if the value of Repository Directory is set to <REMOTE> and the value of BrokerType is ICS.
ControllerTraceLevel	0 through 5	0	Dynamic	This property is valid only if the value of RepositoryDirectory is set to <REMOTE> and the value of BrokerType is ICS.
DeliveryQueue	Any valid JMS queue name	<CONNECTORNAME> /DELIVERYQUEUE	Component restart	This property is valid only when the value of Delivery Transport is JMS.
DeliveryTransport	MQ, IDL, or JMS	IDL when the value of RepositoryDirectory is <REMOTE>, otherwise JMS	Component restart	If the value of RepositoryDirectory is not <REMOTE>, the only valid value for this property is JMS.
DuplicateEventElimina tion	true or false	false	Component restart	This property is valid only if the value of DeliveryTransport is JMS.
EnableOidForFlowMoni toring	true or false	false	Component restart	This property is valid only if the value of BrokerType is ICS.
FaultQueue	Any valid queue name.	<CONNECTORNAME> /FAULTQUEUE	Component restart	This property is valid only if the value of DeliveryTransport is JMS.
jms.FactoryClassName	CxCommon.Messaging.jms .IBMMQSeriesFactory, CxCommon.Messaging .jms.SonicMQFactory, or any Java class name	CxCommon.Messaging. jms.IBMMQSeriesFactory	Component restart	This property is valid only if the value of DeliveryTransport is JMS.

Table 15. Summary of standard configuration properties (continued)

Property name	Possible values	Default value	Update method	Notes
jms.ListenerConcurrency	1 through 32767	1	Component restart	This property is valid only if the value of jms.TransportOptimized is `true`.
jms.MessageBrokerName	If the value of jms.FactoryClassName is IBM, use `crossworlds.queue. manager.`	`crossworlds.queue. manager`	Component restart	This property is valid only if the value of DeliveryTransport is `JMS`.
jms.NumConcurrent Requests	Positive integer	10	Component restart	This property is valid only if the value of DeliveryTransport is `JMS`.
jms.Password	Any valid password		Component restart	This property is valid only if the value of DeliveryTransport is `JMS`.
jms.TransportOptimized	`true` or `false`	`false`	Component restart	This property is valid only if the value of DeliveryTransport is `JMS` and the value of BrokerType is ICS.
jms.UserName	Any valid name		Component restart	This property is valid only if the value of Delivery Transport is `JMS`.
JvmMaxHeapSize	Heap size in megabytes	128m	Component restart	This property is valid only if the value of Repository Directory is set to <REMOTE> and the value of BrokerType is ICS.
JvmMaxNativeStackSize	Size of stack in kilobytes	128k	Component restart	This property is valid only if the value of Repository Directory is set to <REMOTE> and the value of BrokerType is ICS.
JvmMinHeapSize	Heap size in megabytes	1m	Component restart	This property is valid only if the value of Repository Directory is set to <REMOTE> and the value of BrokerType is ICS.
ListenerConcurrency	1 through 100	1	Component restart	This property is valid only if the value of DeliveryTransport is `MQ`.
Locale	This is a subset of the supported locales: `en_US, ja_JP, ko_KR, zh_CN, zh_TW, fr_FR, de_DE, it_IT, es_ES, pt_BR`	`en_US`	Component restart	

Table 15. Summary of standard configuration properties (continued)

Property name	Possible values	Default value	Update method	Notes
LogAtInterchangeEnd	`true` or `false`	`false`	Component restart	This property is valid only if the value of Repository Directory is set to <REMOTE> and the value of BrokerType is ICS.
MaxEventCapacity	1 through 2147483647	`2147483647`	Dynamic	This property is valid only if the value of Repository Directory is set to <REMOTE> and the value of BrokerType is ICS.
MessageFileName	Valid file name	`InterchangeSystem.txt`	Component restart	
MonitorQueue	Any valid queue name	`<CONNECTORNAME>` `/MONITORQUEUE`	Component restart	This property is valid only if the value of DuplicateEventElimination is true and ContainerManagedEvents has no value.
OADAutoRestartAgent	`true` or `false`	`false`	Dynamic	This property is valid only if the value of Repository Directory is set to <REMOTE> and the value of BrokerType is ICS.
OADMaxNumRetry	A positive integer	`1000`	Dynamic	This property is valid only if the value of Repository Directory is set to <REMOTE> and the value of BrokerType is ICS.
OADRetryTimeInterval	A positive integer in minutes	`10`	Dynamic	This property is valid only if the value of Repository Directory is set to <REMOTE> and the value of BrokerType is ICS.
PollEndTime	HH = 0 through 23 MM = 0 through 59	`HH:MM`	Component restart	
PollFrequency	A positive integer (in milliseconds)	`10000`	Dynamic if broker is ICS; otherwise Component restart	
PollQuantity	1 through 500	`1`	Agent restart	This property is valid only if the value of ContainerManagedEvents is JMS.
PollStartTime	HH = 0 through 23 MM = 0 through 59	`HH:MM`	Component restart	
RepositoryDirectory	<REMOTE> if the broker is ICS; otherwise any valid local directory.	For ICS, the value is set to <REMOTE> For WMQI and WAS, the value is `<ProductDir` `\repository`	Agent restart	

Table 15. Summary of standard configuration properties (continued)

Property name	Possible values	Default value	Update method	Notes
RequestQueue	Valid JMS queue name	`<CONNECTORNAME>` /REQUESTQUEUE	Component restart	This property is valid only if the value of DeliveryTransport is JMS
ResponseQueue	Valid JMS queue name	`<CONNECTORNAME>` /RESPONSEQUEUE	Component restart	This property is valid only if the value of DeliveryTransport is JMS.
RestartRetryCount	0 through 99	3	Dynamic if ICS; otherwise Component restart	
RestartRetryInterval	A value in minutes from 1 through `2147483647`	1	Dynamic if ICS; otherwise Component restart	
ResultsSetEnabled	`true` or `false`	`false`	Component restart	Used only by connectors that support DB2II. This property is valid only if the value of DeliveryTransport is JMS, and the value of BrokerType is WMQI.
ResultsSetSize	Positive integer	0 (means the results set size is unlimited)	Component restart	Used only by connectors that support DB2II. This property is valid only if the value of ResultsSetEnabled is `true`.
RHF2MessageDomain	`mrm` or `xml`	`mrm`	Component restart	This property is valid only if the value of DeliveryTransport is JMS and the value of WireFormat is CwXML.
SourceQueue	Any valid WebSphere MQ queue name	`<CONNECTORNAME>` /SOURCEQUEUE	Agent restart	This property is valid only if the value of ContainerManagedEvents is JMS.
SynchronousRequest Queue	Any valid queue name.	`<CONNECTORNAME>` /SYNCHRONOUSREQUEST QUEUE	Component restart	This property is valid only if the value of DeliveryTransport is JMS.
SynchronousRequest Timeout	0 to any number (milliseconds)	0	Component restart	This property is valid only if the value of DeliveryTransport is JMS.
SynchronousResponse Queue	Any valid queue name	`<CONNECTORNAME>` /SYNCHRONOUSRESPONSE QUEUE	Component restart	This property is valid only if the value of DeliveryTransport is JMS.
TivoliMonitorTransaction Performance	`true` or `false`	`false`	Component restart	

Table 15. Summary of standard configuration properties (continued)

Property name	Possible values	Default value	Update method	Notes
WireFormat	CwXML or CwBO	CwXML	Agent restart	The value of this property must be CwXML if the value of RepositoryDirectory is not set to <REMOTE>. The value must be CwBO if the value of RepositoryDirectory is set to <REMOTE>.
WsifSynchronousRequest Timeout	0 to any number (milliseconds)	0	Component restart	This property is valid only if the value of BrokerType is WAS.
XMLNameSpaceFormat	short or long	short	Agent restart	This property is valid only if the value of BrokerType is WMQI or WAS

Standard properties

This section describes the standard connector configuration properties.

AdapterHelpName

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

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

AdminInQueue

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

The default value is *<CONNECTORNAME>*/ADMININQUEUE

AdminOutQueue

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

The default value is *<CONNECTORNAME>*/ADMINOUTQUEUE

AgentConnections

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

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

The default value of this property is 1.

AgentTraceLevel

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

The default value is 0.

ApplicationName

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

The default is the name of the connector.

BiDi.Application

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

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

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

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

BiDi.Broker

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

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

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

BiDi.Metadata

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

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

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

BiDi.Transformation

The BiDi.Transformation property defines whether the system performs a bidirectional transformation at run time.

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

The default value is `false`.

BrokerType

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

CharacterEncoding

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

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

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

CommonEventInfrastructure

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

The default value is `false`.

CommonEventInfrastructureContextURL

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

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

The default value is a blank field.

ConcurrentEventTriggeredFlows

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

Setting this property to a value greater than 1 allows a connector for a source application to map multiple event business objects at the same time and deliver

them to multiple collaboration instances simultaneously. This speeds delivery of business objects to the integration broker, particularly if the business objects use complex maps. Increasing the arrival rate of business objects to collaborations can improve overall performance in the system.

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

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

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

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

The default value is 1.

ContainerManagedEvents

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

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

- PollQuantity = 1 to 500
- SourceQueue = /SOURCEQUEUE

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

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

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

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

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

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

There is no default value.

ControllerEventSequencing

The ControllerEventSequencing property enables event sequencing in the connector controller.

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

The default value is true.

ControllerStoreAndForwardMode

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

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

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

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

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

The default value is true.

ControllerTraceLevel

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

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

The default value is 0.

DeliveryQueue

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

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

The default value is <CONNECTORNAME>/DELIVERYQUEUE.

DeliveryTransport

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

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

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

The default value is JMS.

WebSphere MQ and IDL

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

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

JMS

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

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

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

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

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

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

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

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

This line restricts heap memory usage to a maximum of 768 MB (3 segments * 256 MB). If the process memory grows larger than this limit, page swapping can occur, which can adversely affect the performance of your system.

- Set the value of the IPCCBaseAddress property to 11 or 12. For more information on this property, see the *System Installation Guide for UNIX*.

DuplicateEventElimination

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

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

The default value is `false`.

EnableOidForFlowMonitoring

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

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

The default value is `false`.

FaultQueue

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

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

jms.FactoryClassName

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

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

jms.ListenerConcurrency

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

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

The default value is 1.

jms.MessageBrokerName

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

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

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

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

jms.NumConcurrentRequests

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

The default value is 10.

jms.Password

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

There is no default value.

jms.TransportOptimized

The jms.TransportOptimized property determines if the WIP (work in progress) is optimized. You must have a WebSphere MQ provider to optimize the WIP. For optimized WIP to operate, the messaging provider must be able to:
1. Read a message without taking it off the queue
2. Delete a message with a specific ID without transferring the entire message to the receiver's memory space
3. Read a message by using a specific ID (needed for recovery purposes)
4. Track the point at which events that have not been read appear.

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

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

The default value is false.

jms.UserName

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

There is no default value.

JvmMaxHeapSize

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

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

The default value is 128m.

JvmMaxNativeStackSize

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

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

The default value is 128k.

JvmMinHeapSize

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

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

The default value is 1m.

ListenerConcurrency

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

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

The default value is 1.

Locale

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

A locale name has the following format:

ll_TT.codeset

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

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

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

The default value is en_US.

LogAtInterchangeEnd

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

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

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

The default value is false.

MaxEventCapacity

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

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

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

The default value is 2147483647.

MessageFileName

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

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

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

The default value is `InterchangeSystem.txt`.

MonitorQueue

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

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

The default value is `<CONNECTORNAME>/MONITORQUEUE`

OADAutoRestartAgent

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

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

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

The default value is `false`.

OADMaxNumRetry

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

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

The default value is `1000`.

OADRetryTimeInterval

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

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

The default value is `10`.

PollEndTime

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

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

If the adapter runtime detects:
• PollStartTime set and PollEndTime not set, or
• PollEndTime set and PollStartTime not set

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

PollFrequency

The PollFrequency property specifies the amount of time (in milliseconds) between the end of one polling action and the start of the next polling action. This is not the interval between polling actions. Rather, the logic is as follows:
• Poll to obtain the number of objects specified by the value of the PollQuantity property.
• Process these objects. For some connectors, this may be partly done on separate threads, which execute asynchronously to the next polling action.
• Delay for the interval specified by the PollFrequency property.
• Repeat the cycle.

The following values are valid for this property:
• The number of milliseconds between polling actions (a positive integer).
• The word `no`, which causes the connector not to poll. Enter the word in lowercase.
• The word `key`, which causes the connector to poll only when you type the letter `p` in the connector Command Prompt window. Enter the word in lowercase.

The default is `10000`.

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

PollQuantity

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

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

An e-mail message is also considered an event. The connector actions are as follows when it is polled for e-mail.
• When it is polled once, the connector detects the body of the message, which it reads as an attachment. Since no data handler was specified for this mime type, it will then ignore the message.

- The connector processes the first BO attachment. The data handler is available for this MIME type, so it sends the business object to Visual Test Connector.
- When it is polled for the second time, the connector processes the second BO attachment. The data handler is available for this MIME type, so it sends the business object to Visual Test Connector.
- Once it is accepted, the third BO attachment should be transmitted.

PollStartTime

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

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

If the adapter runtime detects:
- PollStartTime set and PollEndTime not set, or
- PollEndTime set and PollStartTime not set

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

RepositoryDirectory

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

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

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

RequestQueue

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

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

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

ResponseQueue

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

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

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

RestartRetryCount

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

The default value is 3.

RestartRetryInterval

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

Possible values for the property range from 1 through 2147483647.

The default value is 1.

ResultsSetEnabled

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

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

The default value is `false`.

ResultsSetSize

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

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

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

RHF2MessageDomain

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

This is an example header:
```
<mcd><Msd>mrm</Msd><Set>3</Set><Type>
Retek_POPhyDesc</Type><Fmt>CwXML</Fmt></mcd>
```

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

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

SourceQueue

The SourceQueue property designates the JMS source queue for the connector framework in support of guaranteed event delivery for JMS-enabled connectors that use a JMS event store. For further information, see "ContainerManagedEvents" on page 102.

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

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

SynchronousRequestQueue

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

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

The default value is `<CONNECTORNAME>/SYNCHRONOUSREQUESTQUEUE`

SynchronousRequestTimeout

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

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

The default value is `0`.

SynchronousResponseQueue

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

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

The default is `<CONNECTORNAME>/SYNCHRONOUSRESPONSEQUEUE`

TivoliMonitorTransactionPerformance

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

The default value is `false`.

WireFormat

The WireFormat property specifies the message format on the transport:

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

WsifSynchronousRequestTimeout

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

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

The default value is 0.

XMLNameSpaceFormat

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

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

The default value is `short`.

Appendix B. Connector specific properties

This appendix 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 16 lists the connector-specific configuration properties for the connector. See the sections that follow for explanations of the properties.

Table 16. 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
DisableCrossReferencing	`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	CHANGE ME	Yes
PollFutureEvents	`true` or `false`	`false`	No
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
UseEventCI	`true` or `false`	`true`	No

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`.

DisableCrossReferencing

Specifies how key values are set. When key values are generated through PeopleCode, set this property to false, which enables cross referencing. When key values are set against the key attributes in the business object definition, set this property to true.

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.

PollFutureEvents

Specifies whether the connector searches for future-dated events to see if they are ready for polling. When this is set to true, the connector polls by first searching for events with a status of 99, indicating that they are set for future polling. When it finds these events, it checks their date against the system date. If the event date is equal to or less than the system date, the connector changes the event status to 0, indicating that the event is ready for pollong.

When this is set to false, the connector does not check for events set for future polling.

The default value is `false`.

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.

UseEventCI

Specifies whether the component interface CWEVENT_CI is used for outbound processing. When this property is set to true, CWEVENT_CL is used for outbound processing. When this is set to false, no outbound processing occurs.

The default value is true.

Appendix C. Delta event propagation

For environments that use PeopleTools versions 8.44 and 8.45, the delta event propagation process provides an alternative to the DeltaUpdate verb operations for updating business objects between PeopleSoft^R and the adapter with small changes. It is an asynchronous process, and may offer greater processing efficiencies over the recursive verb operations in some circumstances. This appendix includes the following topics:

- "Overview"
- "Process dependencies" on page 120
- "Configuring WebSphere MQ server" on page 120
- "Verifying the WebSphere MQ Client installation" on page 123.
- "Generating the JNDI bindings file" on page 123
- "Configuring PeopleSoft" on page 124

Overview

The delta event propagation process uses the PeopleSoft integration broker, the WebSphere MQ message queue, and the XML ODA and XML data handler, to send business updates from the PeopleSoft application to the adapter. Event polling is never performed as a part of this process.
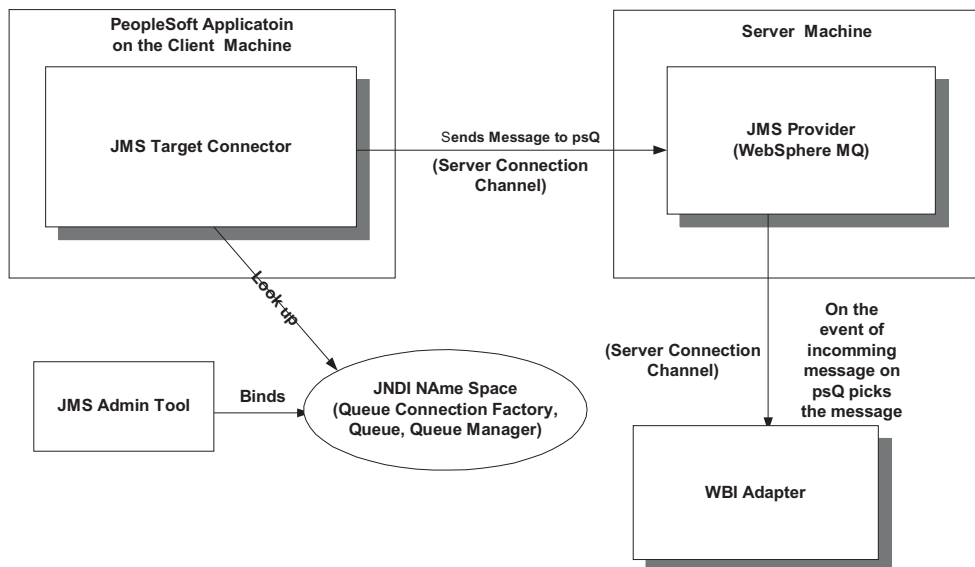
Figure 16. Delta event propagation architecture

The following steps occur during the process:

- You create an XML message definition for each component of interest. The message structure mirrors the structure of the component.
- When you make and save changes to a component, the changes are copied to the corresponding XML message definition instance.

119

- The PeopleSoft integration broker uses the JMSTargetConnector to publish the XML messages onto the MQ message queue.
- The adapter for PeopleSoft picks up the messages from the MQ message queue.
- The adapter uses the XML ODA and the XML data handler to construct a WebSphere business object from the XML message. The business object reflects the changes that were made to the component within the PeopleSoft application.

Process dependencies

This appendix assumes that you are experienced with PeopleSoft component development and application configuration, and with WebSphere MQ installation and configuration. Also, it assumes that you have access to full product documentation for both products. Before you attempt to implement this process, make sure that the following dependencies are satisfied:

- PeopleTools 8.44 or 8.45 and any appropriate patches are installed.
- WebSphere MQ server, version 5.3, is installed on a server machine.
- WebSphere MQ client is installed on the same machine as the PeopleSoft Application.
- You have set the standard property `ContainerManagedEvents` to `true`, and you have specified the `SourceQueue` as the name of the WebSphere MQ message queue where the XML message from PeopleSoft will be routed.

  For more information on setting this property, see "ContainerManagedEvents" on page 102.
- The XML ODA is installed, and the XML data handler is configured.

  When configuring the XML data handler, only the BOPrefix default value will need to be set in the corresponding meta object's attribute.

  For more information, see the *Data Handler Guide.*

Configuring WebSphere MQ server

On the WebSphere MQ server machine, you must configure the following:
- a Queue Manager
- a Queue
- If you are working in a distributed environment: a Server Connection Channel

Configuring a Queue manager

Follow these steps to configure a queue manager.

1. In the WebSphere MQ client, navigate to the Create Queue Manager-Step 1 page. Set the fields as shown:

Field	Setting
Queue Manager	psQM
Dead Letter Queue	SYSTEM.DEAD.LETTER.QUEUE
Max Handle Limit	256
Trigger Interval	999999999
Uncommitted Mssgs	1000

2. Click Next to open the Create Queue Manager (Step 2) page. page. Set the fields as shown:

Field	Setting
Use Circular Logging	Selected
Log Path	C:\program files\ibm\websphere mq\log
Log File Size	256
Log Primary Files	3
Log Secondary Files	2

3. Click `Next`to open the Create Queue Manager (Step 3) page.
4. Select the `Start Queue Manager` check box
5. Click `Next`to open the Create Queue Manager (Step 4) page.
6. Select the `Create listener configured for TCP/IP` check box.
7. At the `Listen on port number:` field, enter

   `81`

   .
8. Click

   `Finish.`

Configuring a queue

Follow these steps to configure a queue.

1. In the MQ client, navigate to Local Queue.
2. On the general tab, select the following settings for the fields shown:

Field	Setting
Queue name	Name of the Queue you are creating, for example, psQ
Description	Websphere Default Local Queue
Put Messages	Allowed
Get Messages	Allowed
Default Priority	0
Default Persistence	Not Persistent
Scope	Queue Manager
Usage	Normal

3. Click OK.
4. On the Extended tab, select the following settings for the fields shown:

Field	Setting
Maximum Queue Length	5000
Maximum Message Length	4194304
Shareability	Shareable
Default Input Open Option	Shared
Message delivery Sequence	Priority
Retention Level	999999999
Distribution Lists	Disabled

5. Click OK.
6. On the Cluster tab, set the following:
   - Select `Not Shared in Cluster.`
   - Select `On Open` at the `Default Bind:` field.
7. Click OK.
8. On the Events tab, select the following settings for the fields shown:

Field	Setting
Maximum Depth Event	Extended
High Depth Event	Disabled
High Depth Limit	80
Low Depth Event	Disabled
Low Depth Limit	20
Service Interval Event	None
Service Interval	999999999

9. Click OK.
10. On the Storage tab, set the following:
    - Set `Backout Threshold` to `0`.
    - Set `Harden Get Backout` to `Harden`.
11. Click OK.

Configuring a server channel

If you are working in a distributed environment, you must configure a server channel. If you are not working in a distributed environment, you do not need to complete this task.

1. Navigate to the Create Server Connection page.
2. On the General tab, set the following:
   - Enter a name for the `Channel Name`.
   - Set `Transmission Protocol` to `TCP/IP`.
3. Click OK.
4. On the Extended tab, set the following:
   - Set `Maximum Message Length` to `4194304`.
   - Set `Heartbeat Interval` to `300`.
5. Click OK.
6. On the MCA tab, enter an `MCA User ID`. Make sure that the user ID you enter is legitimate on the operating system you use, and on both the server and client machines.
7. Click OK.
8. On the SSL tab, select `none` at the `Standard settings` field.

Setting up the class path for the WebSphere MQ client jars

You must set up a class path for the WebSphere MQ client jars. Follow these steps to do so:

1. Log on to WebSphere Application Server Admin Console.

2. From the left navigation pane, click Servers > Application Servers > <Server Name> where the <Server Name> is the name of the server on the right hand pane.
3. Under Additional Properties, click Process Definition.
4. Under Additional Properties, click the Java Virtual Machine.
5. In the Class Path, set the following:

```
<MQ_Client>/lib;<MQ_Client>/lib/providerutil.jar;
<MQ_Client>/lib/com.ibm.mq.jar;<MQ_Client>/lib/com.ibm.mqbind.jar;
<MQ_Client>/lib/jms.jar;<MQ_Client>/lib/jndi.jar;
<MQ_Client>/lib/com.ibm.mqjms.jar;<MQ_Client>/lib/fscontext.jar;
<MQ_Client>/lib/connector.jar;<MQ_Client>/lib/jta.jar;
<MQ_Client>/lib/postcard.jar;
```

Where <MQ_Client> is the installed directory of the WebSphere MQ Client.
6. Click OK and then click Save to save the change.
7. Reboot the WebSphere Application Server.

Verifying the WebSphere MQ Client installation

Verify the Websphere MQ client is installed on the same machine as your PeopleSoft application. Note the directory where the MQ Client files (com.ibm.mq.jar, com.ibm.mqbind.jar etc.) are installed, and create a subdirectory under this directory. Label the subdirectory `bin`.

Generating the JNDI bindings file

You must generate a JNDI bindings file on the server machine, and then transfer it to the \bin subdirectory on the client machine.

1. On the server machine, create a directory and label it `C:\JNDI-Directory`. If this directory already exists, delete any bindings files from it.
2. On the server machine, navigate to the java\bin directory of WebSphere MQ and make the following change in the JMSAdmin config file

```
INITIAL_CONTEXT_FACTORY=com.sun.jndi.fscontext.
RefFSContextFactory PROVIDER_URL=file:/C:/JNDI-Directory
```

3. Review the pssetup.bat file or ivtsetup.bat file.
   - Set the correct class path.
   - Verify that qcf and q JMS administered JNDI objects point to the correct Queue Manager, Queue, and Server Connection Channel, if needed, of WebSphere MQ Server.

     The following code provides an example of what you should see in the file.

```
set CLASSPATH=C:\Program Files\IBM\WebSphere MQ\Java\lib\com.ibm.mqjms.jar;
C:\Program Files\IBM\WebSphere MQ\Java\lib\com.ibm.mq.jar;
C:\Program Files\IBM\WebSphere MQ\Java\lib\jms.jar;
C:\Program Files\IBM\WebSphere MQ\Java\lib\jndi.jar;
C:\Program Files\IBM\WebSphere MQ\Java\lib\providerutil.jar;
C:\Program Files\IBM\WebSphere MQ\Java\lib\ldap.jar;
C:\Program Files\IBM\WebSphere MQ\Java\lib\fscontext.jar;
C:\Program Files\IBM\WebSphere MQ\Java\lib\connector.jar


echo + Creating script for object creation within JMSAdmin
rem echo def qcf(psQCF) TRANSPORT(CLIENT) > ivtsetup.scp
rem the following line to be used in place of above for mq  client scenario
echo def qcf(psQCF) TRANSPORT(CLIENT) HOST(SMEKA) PORT(1415)
CHANNEL(ps_SRV_CHANNEL) QMGR(psQM) > ivtsetup.scp
echo def q(psQ) qu(psQ) >> ivtsetup.scp
```

```
        echo end >> ivtsetup.scp

        echo + Calling JMSAdmin in batch mode to create objects
        java -DMQJMS_LOG_DIR="%MQ_JAVA_INSTALL_PATH%"\
        log -DMQJMS_TRACE_DIR="%MQ_JAVA_INSTALL_PATH%"\
        trace -DMQJMS_INSTALL_PATH="%MQ_JAVA_INSTALL_PATH%"
         com.ibm.mq.jms.admin.JMSAdmin < ivtsetup.scp

        echo + Administration done; tidying up files
        del ivtsetup.scp

        echo + Done!
        PAUSE
```

4. Run the bat file from the command line in java/bin directory. This creates a.bindings file in the C:\JNDI-Directory.

5. Copy this file to the client machine to the bin directory of the WebSphere MQ Client Installation.

Configuring PeopleSoft

You must complete some initial configuration tasks within PeopleSoft, and then some additional configuration tasks required for each component.

Note: For more information about the tasks listed in these sections, refer to your PeopleTools documentation that describes the PeopleSoft Integration Broker.

Configuring the PeopleSoft application

Complete the following tasks. These are required to set the PeopleSoft application correctly for the delta update propagation process.

1. Activate the Pub/Sub Server Domain and add a node definition.

   Make sure that the following values are set for the node properties.

Property	Value
Node Name	User created
Node Description	User created
Node Type	PIA
Routing	Implicit
Authentication	None
Activate Node check box	Selected

2. Specify a gateway and connector for the current node. If the gateway is not available, create one locally with a URL similar to this example:

   `http://gateway_server/PSIGW/PeopleSoftListeningConnector`

   Note: Gateway_server is the machine name and port, host name, or IP address of the web server hosting the gateway. The gateway uses the PeopleSoft listening connector to receive messages from an integration engine node or a remote gateway.

3. Edit the JMSTargetConnector properties so that their values match those shown below.

Property	Value	Notes
sendUmcompressed	Y	

Property	Value	Notes
JMSAcknowledgement	AUTO_ACKNOWLEDGE	
JMSDeliveryMode	NON_PERSISTENT	
JMSFactory	Example value: psQCF	Make this the value of the JMS administered QueueConnectionFactory object in the.bat file.
JMSMessageTimeToLive	0	
JMSMessageType	Text	
JMSPriority	0	
JMSProvider	MQSeries	
JMSQueue	Example value: psQ	Add this property if it does not already exist. Make its value the value of the JMS administered Queue object in the.bat file.
JMSReplyTo	FALSE	
JMSUrl		Point to the directory where you put the.bindings file on the client machine.

4. Use PeopleTools Application Designer to create a project.

Configuring PeopleSoft components

In addition to the initial tasks you must complete to set up the PeopleSoft application for delta event propagation, you must also complete the following tasks for each component for which you want delta events propagated.

1. Use PeopleTools Application Designer to build the component and to include the component in the project.
2. Define a message with a structure similar to the component of interest.

   This means that the message must have the same hierarchy of records as the component. Make sure all primary records are included, and if the component has up datable views, make sure the view records are also included in the message structure.
3. Open the SavePostChange event function for the component in Application Designer. Add the following PeopleCode at the beginning of the file, replacing TEST_MSG with the name of the message you have just created in step 2.

```
Local Message &MSG;
Local Rowset &ComponentBuffer;

If ComponentChanged() Then
     &ComponentBuffer = GetLevel0();

   &MSG = CreateMessage(Message.TEST_MSG);
   &MSG.CopyRowsetDelta(&ComponentBuffer);

   &MSG.Publish();
End-If;
```

4. Add an Asynchronous Outbound Transaction to the node.
5. Generate a WSDL document for the message created in step 2. When the WSDL is generated, it will open in a browser.
6. Select all in your browser and save as a.WSDL file.

7. Use the following script to extract the xsd file from the WSDL file you just saved.

```
sed '/wsdl/d' $1 | sed '/soap/d' | sed '/xml
version="1.0"/d'| sed 's/- <xsd:/  <xsd:/g' | sed 's/-
<\/xsd:/<\/xsd:/g' | sed 's/  <xsd:/<xsd:/g' | sed 's/
<\/xsd:/<\/xsd:/g' > $2
```

Save the script to a file, named convertWsdlToXSD.sh. The script executes as follows:

```
sh convertWsdlToXSD.sh <INPUTFILENAME> <OUTPUTFILENAME>
where <INPUTFILENAME> is the name of the WSDL file and
<OUTPUTFILENAME> is the name of the XSD file
to be generated
```

You must complete steps 1 through 7 for each PeopleSoft component for which you want delta changes to be routed from PeopleSoft to the adapter.

Use the XMLODA to generate the business object definition for the corresponding XSD extracted in step 5 above. This definition will be used by the adapter to send the events to the integration broker, WebSphere InterChange Server.

Appendix D. Common Event Infrastructure

WebSphere Business Integration Server Foundation includes the Common Event Infrastructure Server Application, which is required for Common Event Infrastructure to operate. The WebSphere Application Server Foundation can be installed on any system (it does not have to be the same machine on which the adapter is installed.)

The WebSphere Application Server Application Client includes the libraries required for interaction between the adapter and the Common Event Infrastructure Server Application. You must install WebSphere Application Server Application Client on the same system on which you install the adapter. The adapter connects to the WebSphere Application Server (within the WebSphere Business Integration Server Foundation) by means of a configurable URL.

Common Event Infrastructure support is available using any integration broker supported with this release.

Required software

In addition to the software prerequisites required for the adapter, you must have the following installed for Common Event Infrastructure to operate:
- WebSphere Business Integration Server Foundation 5.1.1
- WebSphere Application Server Application Client 5.0.2, 5.1, or 5.1.1.

  (WebSphere Application Server Application Client 5.1.1 is provided with WebSphere Business Integration Server Foundation 5.1.1.)

Note: Common Event Infrastructure is not supported on any HP-UX or Linux platform.

Enabling Common Event Infrastructure

Common Event Infrastructure functionality is enabled with the standard properties `CommonEventInfrastructure` and `CommonEventInfrastructureContextURL`, configured with Connector Configurator. By default, Common Event Infrastructure is not enabled. The `CommonEventInfrastructureContextURL` property enables you to configure the URL of the Common Event Infrastructure server.(Refer to the "Standard Properties" appendix of this document for more information.)

Obtaining Common Event Infrastructure adapter events

If Common Event Infrastructure is enabled, the adapter generates Common Event Infrastructure events that map to the following adapter events:
- Starting the adapter
- Stopping the adapter
- An application response to a timeout from the adapter agent
- Any `doVerbFor` call issued from the adapter agent
- A `gotApplEvent` call from the adapter agent

For another application (the "consumer application") to receive the Common Event Infrastructure events generated by the adapter, the application must use the

Common Event Infrastructure event catalog to determine the definitions of appropriate events and their properties. The events must be defined in the event catalog for the consumer application to be able to consume the sending application's events.

The "Common Event Infrastructure event catalog definitions" appendix of this document contains XML format metadata showing, for WebSphere Business Information adapters, the event descriptors and properties the consumer application should search for.

For more information

For more information about Common Event Infrastructure, refer to the Common Event Infrastructure information in the WebSphere Business Integration Server Foundation documentation, available at the following URL:

http://publib.boulder.ibm.com/infocenter/ws51help

For sample XML metadata showing the adapter-generated event descriptors and properties a consumer application should search for, refer to"Common Event Infrastructure event catalog definitions."

Common Event Infrastructure event catalog definitions

The Common Event Infrastructure event catalog contains event definitions that can be queried by other applications. The following are event definition samples, using XML metadata, for typical adapter events. If you are writing another application, your application can use event catalog interfaces to query against the event definition. For more information about event definitions and how to query them, refer to the Common Event Infrastructure documentation that is available from the online IBM WebSphere Server Foundation Information Center.

For WebSphere Business Integration adapters, the extended data elements that need to be defined in the event catalog are the keys of the business object. Each business object key requires an event definition. So for any given adapter, various events such as start adapter, stop adapter, timeout adapter, and any `doVerbFor` event (create, update, or delete, for example) must have a corresponding event definition in the event catalog.

The following sections contain examples of the XML metadata for start adapter, stop adapter, and event request or delivery.

XML format for "start adapter" metadata

```
<eventDefinition name="startADAPTER"
      parent="event">
   <property name ="creationTime" //Comment: example value would be
"2004-05-13T17:00:16.319Z"
         required="true" />
   <property name="globalInstanceId" //Comment: Automatically generated
 by Common Event Infrastructure
         required="true"/>
   <property name="sequenceNumber"    //Comment: Source defined number
for messages to be sent/sorted logically
         required="false"/>
   <property name="version"    //Comment: Version of the event
         required="false"
         defaultValue="1.0.1"/>
```

```
        <property name="sourceComponentId"
             path="sourceComponentId"
             required="true"/>
        <property name="application"    //Comment: The name#version of the
 source application generating the event. Example is "SampleConnector#3.0.0"
             path="sourceComponentId/application"           required="false"/>
        <property name="component"    //Comment: This will be the name#version
 of the source component.
             path="sourceComponentId/component"
             required="true"
             defaultValue="ConnectorFrameWorkVersion#4.2.2"/>
        <property name="componentIdType"     //Comment: specifies the format
 and meaning of the component
             path="sourceComponentId/componentIdType"
             required="true"
             defaultValue="Application"/>
        <property name="executionEnvironment"
 //Comment: Identifies the environment the application is running
 in...example is "Windows 2000#5.0"
             path="sourceComponentId/executionEnvironment"
             required="false" />
         <property name="location"     //Comment: The value of this is the
 server name...example is "WQMI"
             path="sourceComponentId/location"
             required="true"/>
        <property name="locationType"    //Comment specifies the format and
     meaning of the location
             path="sourceComponentId/locationType"
             required="true"
             defaultValue="Hostname"/>
        <property name="subComponent"       //Comment:further distinction
 of the logical component
             path="sourceComponentId/subComponent"
             required="true"
             defaultValue="AppSide_Connector.AgentBusinessObjectManager"/>
        <property name="componentType"        //Comment: well-defined name
 used to characterize all instances of this component
             path="sourceComponentId/componentType"
             required="true"
             defaultValue="ADAPTER"/>
        <property name="situation"    //Comment: Defines the type of
 situation that caused the event to be reported
             path="situation"
             required="true"/>
        <property name="categoryName="     //Comment: Specifies the type
 of situation for the event
             path="situation/categoryName"
             required="true"
             defaultValue="StartSituation"/>
        <property name="situationType"     //Comment: Specifies the type
 of situation and disposition of the event
             path="situation/situationType"
             required="true"
        <property name="reasoningScope" //Comment: Specifies the scope
 of the impact of the event
             path="situation/situationType/reasoningScope"
             required="true"
             permittedValue="INTERNAL"
             permittedValue="EXTERNAL"/>
        <property name="successDisposition" //Comment: Specifies the
 success of event
             path="situation/situationType/successDisposition"
             required="true"
             permittedValue="SUCCESSFUL"
             permittedValue="UNSUCCESSFUL" />
        <property name="situationQualifier"  //Comment: Specifies the
 situation qualifiers for this event
```

```
                  path="situation/situationType/situationQualifier"
                  required="true"
                  permittedValue="START_INITIATED"
                  permittedValue="RESTART_INITIATED"
                  permittedValue="START_COMPLETED" />
            </eventDefinition>
```

XML format for ″stop adapter″ metadata

The metadata for "stop adapter" is the same as that for "start adapter" with the
following exceptions:

- The default value for the categoryName property is StopSituation:

```
<property name="categoryName="
 //Comment: Specifies the type
 of situation for the event
              path="situation/categoryName"
              required="true"
              defaultValue="StopSituation"/>
```

- The permitted values for the situationQualifier property differ and are as
  follows for "stop adapter":

```
<property name="situationQualifier"
 //Comment: Specifies the situation qualifiers for this event
            path="situation/situationType/situationQualifier"
            required="true"
            permittedValue="STOP_INITIATED"
            permittedValue="ABORT_INITIATED"
            permittedValue="PAUSE_INITIATED"
            permittedValue="STOP_COMPLETED"
     />
```

XML format for "timeout adapter" metadata

The metadata for "timeout adapter" is the same as that for "start adapter" and
"stop adapter" with the following exceptions:

- The default value for the categoryName property is ConnectSituation:

```
<property name="categoryName="
 //Comment: Specifies the type
 of situation for the event
            path="situation/categoryName"
            required="true"
            defaultValue="ConnectSituation"/>
```

- The permitted values for the situationQualifier property differ and are as
  follows for "timeout adapter":

```
<property name="situationQualifier"  //Comment: Specifies
 the situation qualifiers for this event
            path="situation/situationType/situationQualifier"
            required="true"
            permittedValue="IN_USE"
            permittedValue="FREED"
            permittedValue="CLOSED"
            permittedValue="AVAILABLE"
     />
```

XML format for ″request″ or ″delivery″ metadata

At the end of this XML format are the extended data elements. The extended data elements for adapter request and delivery events represent data from the business object being processed. This data includes the name of the business object, the key (foreign or local) for the business object, and business objects that are children of parent business objects. The children business objects are then broken down into the same data as the parent (name, key, and any children business objects). This data is represented in an extended data element of the event definition. This data will change depending on which business object, which keys, and which child business objects are being processed. The extended data in this event definition is just an example and represents a business object named Employee with a key EmployeeId and a child business object EmployeeAddress with a key EmployeeId. This pattern could continue for as much data as exists for the particular business object.

```
<eventDefinition name="createEmployee"      //Comment: This
 extension name is always the business object verb followed by the business
 object name
          parent="event">
    <property name ="creationTime"  //Comment: example value would be
"2004-05-13T17:00:16.319Z"
          required="true" />
    <property name="globalInstanceId" //Comment: Automatically generated
 by Common Event Infrastructure
          required="true"/>
    <property name="localInstanceId"     //Comment: Value is business
 object verb+business object name+#+app name+ business object identifier
          required="false"/>
    <property name="sequenceNumber"     //Comment: Source defined number
 for messages to be sent/sorted logically
          required="false"/>
    <property name="version"  //Comment: Version of the event...value is
 set to 1.0.1
          required="false"
          defaultValue="1.0.1"/>
    <property name="sourceComponentId"
          path="sourceComponentId"
          required="true"/>
    <property name="application"     //Comment: The name#version of the
 source application generating the event...example is
"SampleConnector#3.0.0"
          path="sourceComponentId/application"
          required="false"/>
    <property name="component"    //Comment: This will be the name#version
 of the source component.
          path="sourceComponentId/component"
          required="true"
          defaultValue="ConnectorFrameWorkVersion#4.2.2"/>
    <property name="componentIdType"      //Comment: specifies the format
 and meaning of the component
          path="sourceComponentId/componentIdType"
          required="true"
          defaultValue="Application"/>
    <property name="executionEnvironment" //Comment: Identifies the
 environment#version the app is running in...example is "Windows 2000#5.0"
          path="sourceComponentId/executionEnvironment"
          required="false" />
    <property name="instanceId" //Comment: Value is business object
  verb+business object name+#+app name+ business object identifier
          path="sourceComponentId/instanceId"
          required="false"
    <property name="location"    //Comment: The value of this is the
 server name...example is "WQMI"
          path="sourceComponentId/location"
```

```
            required="true"/>
    <property name="locationType" //Comment specifies the format and
meaning of the location
            path="sourceComponentId/locationType"
            required="true"
            defaultValue="Hostname"/>
    <property name="subComponent"  //Comment:further distinction of the
logical component-in this case the value is the name of the business
object
            path="sourceComponentId/subComponent"
            required="true"/>
    <property name="componentType"      //Comment: well-defined name used
to characterize all instances of this component
            path="sourceComponentId/componentType"
            required="true"
            defaultValue="ADAPTER"/>
    <property name="situation" //Comment: Defines the type of
situation that caused the event to be reported
            path="situation"
            required="true"/>
    <property name="categoryName"     //Comment: Specifies the type
of situation for the event
            path="situation/categoryName"
            required="true"
            permittedValue="CreateSituation"
            permittedValue="DestroySituation"
            permittedValue="OtherSituation" />
    <property name="situationType"     //Comment: Specifies the type
of situation and disposition of the event
            path="situation/situationType"
            required="true"
    <property name="reasoningScope" //Comment: Specifies the scope
of the impact of the event
            path="situation/situationType/reasoningScope"
            required="true"
            permittedValue="INTERNAL"
            permittedValue="EXTERNAL"/>
    <property name="successDisposition" //Comment: Specifies the
success of event
            path="situation/situationType/successDisposition"
            required="true"
            permittedValue="SUCCESSFUL"
            permittedValue="UNSUCCESSFUL" />
    <extendedDataElements name="Employee" //Comment: name of business
object itself
              type="noValue"
              <children name="EmployeeId"
                     type="string"/>  //Comment: type is one of the
permitted values within Common Event Infrastructure documentation
              <children name="EmployeeAddress"
                     type="noValue"/>
                        <children name="EmployeeId"
                            type="string"/>
                      -
                      -
                      -
    </extendedDataElements>
</eventDefinition>
```

Appendix E. Application Response Management

This adapter is compatible with the Application Response Measurement application programming interface (API), an API that allows applications to be managed for availability, service level agreements, and capacity planning. An ARM-instrumented application can participate in IBM Tivoli Monitoring for Transaction Performance, allowing collection and review of data concerning transaction metrics.

Application Response Measurement instrumentation support

This adapter is compatible with the Application Response Measurement application programming interface (API), an API that allows applications to be managed for availability, service level agreements, and capacity planning. An ARM-instrumented application can participate in IBM Tivoli Monitoring for Transaction Performance, allowing collection and review of data concerning transaction metrics.

Required software

In addition to the software prerequisites required for the adapter, you must have the following installed for ARM to operate:

- WebSphere Application Server 5.0.1 (contains the IBM Tivoli Monitoring for Transaction Performance server). This does not have to be installed on the same system as the adapter.
- IBM Tivoli Monitoring for Transaction Performance v. 5.2 Fixpack 1. This must be installed on the same system on which the adapter is installed and configured to point to the system on which the IBM Tivoli Monitoring for Transaction Performance server resides.

Application Response Measurement support is available using any integration broker supported with this release.

Note: Application Response Measurement instrumentation is supported on all operating systems supported with this IBM WebSphere Business Integration Adapters release *except* HP-UX (any version) and Red Hat Linux 3.0.

Enabling Application Response Measurement

ARM instrumentation is enabled via by setting the standard property `TivoliMonitorTransactionPerformance` in Connector Configurator to "True." By default ARM support is not enabled. (Refer to the "Standard Properties" appendix of this document for more information.)

Transaction monitoring

When ARM is enabled, the transactions that are monitored are service events and event deliveries. The transaction is measured from the start of a service request or event delivery to the end of the service request or event delivery. The name of the transaction displayed on the Tivoli Monitoring for Transaction Performance console will start with either `SERVICE REQUEST` or `EVENT DELIVERY`. The next part of the name will be the business object verb (such as `CREATE`, `RETRIEVE`, `UPDATE` or `DELETE`). The final part of the name will be the business object name such as "EMPLOYEE."

For example, the name of a transaction for an event delivery for creation of an employee might be EVENT DELIVERY CREATE EMPLOYEE. Another might be SERVICE REQUEST UPDATE ORDER.

The following metrics are collected by default for each type of service request or event delivery:

- Minimum transaction time
- Maximum transaction time
- Average transaction time
- Total transaction runs

You (or the system administrator of the WebSphere Application Server) can select which of these metrics to display, for which adapter events, by configuring Discovery Policies and Listener Policies for particular transactions from within the Tivoli Monitoring for Transaction Performance console. (Refer to "For more information.")

For more information

Refer to the IBM Tivoli Monitoring for Transaction Performance documentation for more information. In particular, refer to the *IBM Tivoli Monitoring for Transaction Performance User's Guide* for information about monitoring and managing the metrics generated by the adapter.

Index

Special characters

Notices

This information was developed for products and services offered in the U.S.A. IBM may not offer the products, services, or features discussed in this document in other countries. Consult your local IBM representative for information on the products and services currently available in your area. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any non-IBM product, program, or service. IBM may have patents or pending patent applications covering subject matter described in this document. The furnishing of this document does not grant you any license to these patents. You can send license inquiries, in writing, to:

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

For license inquiries regarding double-byte (DBCS) information, contact the IBM Intellectual Property Department in your country or send inquiries, in writing, to:

IBM World Trade Asia Corporation Licensing
2-31 Roppongi 3-chome, Minato-ku
Tokyo 106-0032, Japan

The following paragraph does not apply to the United Kingdom or any other country where such provisions are inconsistent with local law: INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Some states do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement may not apply to you. This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time without notice. Any references in this information to non-IBM Web sites are provided for convenience only and do not in any manner serve as an endorsement of those Web sites. The materials at those Web sites are not part of the materials for this IBM product and use of those Web sites is at your own risk. IBM may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation to you. Licensees of this program who wish to have information about it for the purpose of enabling: (i) the exchange of information between independently created programs and other programs (including this one) and (ii) the mutual use of the information which has been exchanged, should contact:

Programming interface information

Programming interface information, if provided, is intended to help you create application software using this program. General-use programming interfaces allow you to write application software that obtain the services of this program's tools. However, this information may also contain diagnosis, modification, and tuning information. Diagnosis, modification and tuning information is provided to help you debug your application software.

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

Trademarks and service marks

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

i5/OS
IBM
the IBM logo
AIX
CICS
CrossWorlds
DB2
DB2 Universal Database
Domino
IMS
Informix
iSeries
Lotus
Lotus Notes
MQIntegrator
MQSeries
MVS
OS/400
Passport Advantage
SupportPac
WebSphere
z/OS

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. Linux is a trademark of Linus Torvalds in the United States, other countries, or both. Other company, product or service names may be trademarks or service marks of others.

IBM WebSphere Business Integration Adapter Framework V2.6.0

IBM ®

Printed in USA