WebSphere Business Integration Express Plus for Item Synchronization

Adapter for mySAP.com (SAP R/3 V. 3.x) User Guide

V4.3.1

WebSphere Business Integration Express Plus for Item Synchronization

Adapter for mySAP.com (SAP R/3 V. 3.x) User Guide

V4.3.1

Note!
Before using this information and the product it supports, read the information in "Notices" on page 317.

6February2004

This edition of this document applies to IBM WebSphere Business Integration Adapter for mySAP.com (SAP R/3 Version 3.x) and to all subsequent releases and modifications until otherwise indicated in new editions.

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 products IBM(R) WebSphere(R) Business Integration Express for Item Synchronization and IBM(R) WebSphere(R) Business Integration Express Plus for Item Synchronization are made up of the following components: InterChange Server Express, the associated Toolset Express product, the Item Synchronization collaboration, and a set of software integration adapters. Together, the components provide business process integration and connectivity among leading e-business technologies and enterprise applications as well as integration with the UCCnet GLOBAL registry.

This document describes the IBM WebSphere InterChange Server connector for SAP R/3 including how to install, configure, develop business objects, test, and manage your custom development.

Except where noted, all the information in this guide applies to both IBM(R) WebSphere(R) Business Integration Express for Item Synchronization and IBM(R) WebSphere(R) Business Integration Express Plus for Item Synchronization. The term "WebSphere Business Integration Express for Item Synchronization" and its variants refer to both products.

Audience

This document is for IBM WebSphere InterChange Server consultants and customers. You should be familiar with SAP and connector development.

Related documents

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

This document contains many references to two other documents: the *System Installation Guide for Windows* or *System Installation Guide for UNIX* and the *System 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 the documentation from the following site:
http://www.ibm.com/websphere/wbiitemsync/express/infocenter

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.

Naming conventions

In this document the following naming conventions are used:

- The IBM WebSphere InterChange Server connector for SAP is referred to simply as the connector.
- The "connector" refers to the combination of the vision connector framework and a connector module.
- All references to names of IBM WebSphere InterChange Server ABAP objects (such as function modules, programs, and tables) are valid for the connector that supports SAP R/3 version 3.x.

New in this release

New in V4.3.1

February 2004

- New appendix with quick configuration steps
- Updated information on the ALE module

December 2003

Version 5.4.2 is the first release as part of the IBM(R) WebSphere(R) Business Integration Express for Item Synchronization and IBM(R) WebSphere(R) Business Integration Express Plus for Item Synchronization release.

Except where noted, all the information in this guide applies to both IBM(R) WebSphere(R) Business Integration Express for Item Synchronization and IBM(R) WebSphere(R) Business Integration Express Plus for Item Synchronization. The term "WebSphere Business Integration Express for Item Synchronization" and its variants refer to both products.

Part 1. Connector overview and setup

Chapter 1. Overview of the connector

This chapter describes the connector component of the IBM WebSphere Business Integration Adapter for mySAP.com (SAP R/3 Version 3.x). The connector enables an integration broker to exchange business objects with SAP applications.

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

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

This document contains information about the application-specific component and connector framework. It refers to both of these components as the connector.

For more information about the relationship of the integration broker to the connector, see the *IBM CrossWorlds System Administration Guide* (if InterChange Server is the integration broker).

This chapter contains the following sections:

Connector components

The connector for SAP is written in Java and consists of two parts: the vision connector framework and connector modules (the connector's application-specific component, the Connector Framework, and business object handlers). The vision connector framework provides a metadata-driven layer of abstraction to the Connector Framework used by all WebSphere business integration system adapters.

The vision connector framework extends the methods in the system-wide Connector Framework. The connector modules extend the methods in the vision connector framework and communicate with an SAP application.

Note: By default, the connector uses the ABAP Extension Module to support the vision connector framework. For more information on the ABAP Extension Module, see Chapter 3, "Overview of the ABAP Extension module," on page 33.

Figure 1 illustrates the architecture of the connector and the relationship of the system-wide and vision connector frameworks. The visionConnector class can implement any number of connector modules.

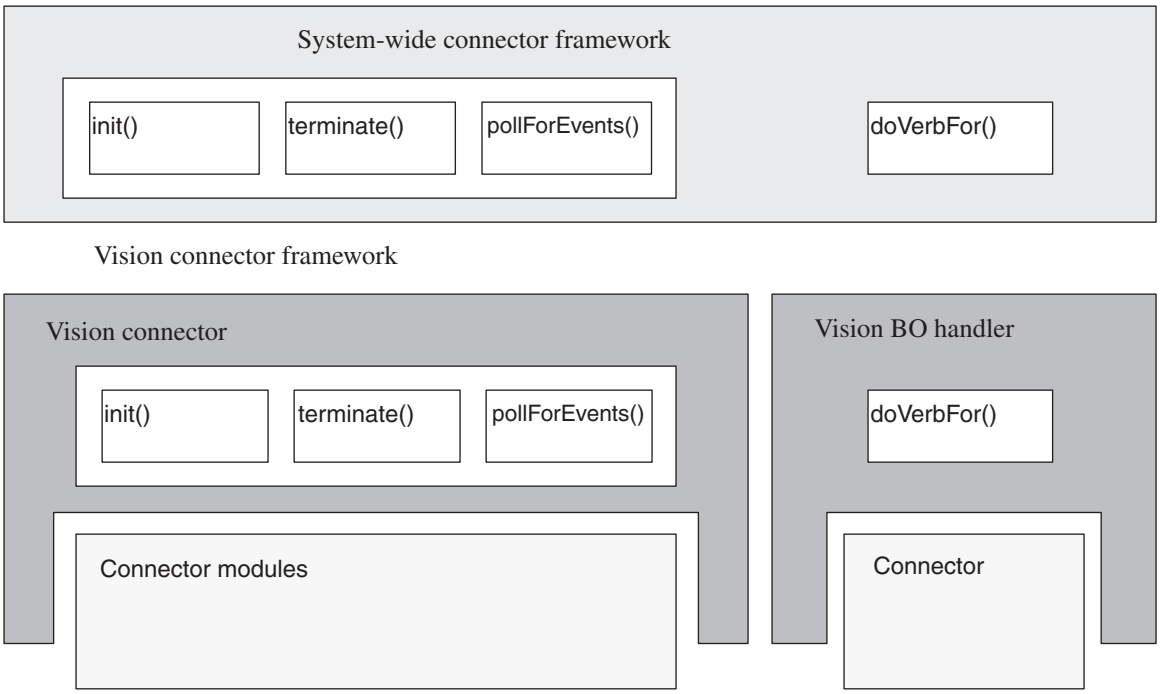


Figure 1. Architecture of the connector for SAP

Vision connector framework

The vision connector framework dynamically routes the initialization, poll, and termination requests to connector modules. It also dynamically routes business objects to business object handlers. A business object handler is a connector module designed specifically to support business objects. To dynamically route requests and business objects, the connector uses the verb application-specific information of a business object and values of certain application-specific connector configuration properties.

The vision connector framework consists of two classes: visionConnector and visionBOHandler.

Figure 2 illustrates the vision connector framework and its association with connector modules.

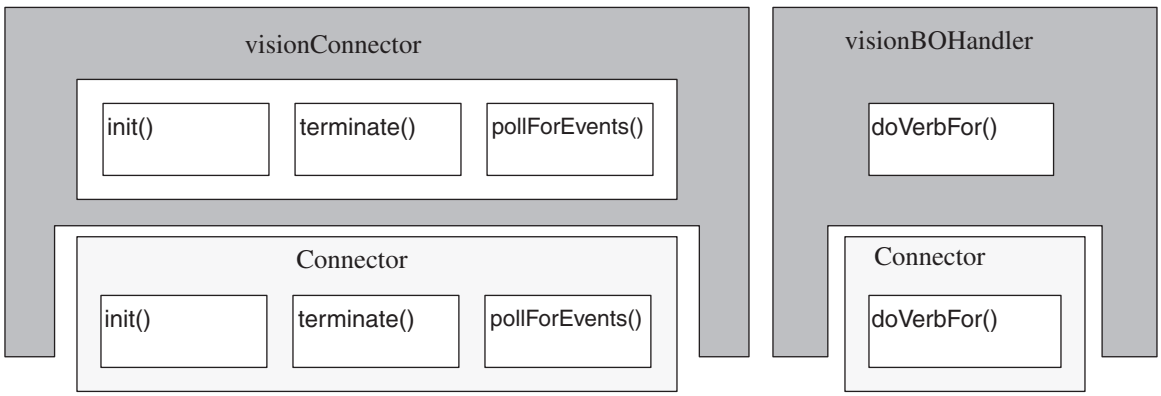


Figure 2. vision connector framework and connector modules

The vision connector framework provides the following capabilities for the connector:

- Calls any implementation of the `init()`, `pollForEvents()`, and `terminate()` methods.
- Routes business objects to specific business object handlers based on the verb application-specific information of a business object.

Connector modules

The connector modules are Java classes that extend the methods in the vision connector framework. They support the vision connector framework by providing specific functionality, such as logging in to the SAP application, processing events and business objects, and terminating the connection to the SAP application. The connector modules carry out requests between the vision connector framework and the SAP application. By default, the vision connector framework uses the `connectors\SAP` directory as the root directory for the connector modules.

Connector modules may not use all of the framework methods. For example, one module might use the `init()` and `terminate()` methods while another module uses only the `pollForEvents()` method. However, every method in the visionConnector and visionBOHandler classes must be implemented for each connector module. Methods that a connector does not use must be implemented as dummy methods, that is, they do nothing but exit.

How the vision connector framework works

The connector interacts with an SAP application using connector modules. The connector modules make calls to SAP's Native Interfaces and pass data (business object or event data) to and from an SAP application. The connector's flexible design enables different modules to be used for different tasks such as initializing the connector with the SAP application or passing business object data.

Communication between the connector and an SAP application

The connector uses SAP's Remote Function Call (RFC) library to communicate with an SAP application. SAP's RFC API allows external programs to call ABAP function modules within an SAP application.

Processing business objects

The connector is metadata driven. metadata, in the WebSphere business integration system, is application-specific data that is stored in business objects and that assists a connector module in its interaction with the application. A metadata-driven connector module handles each business object that it supports based on metadata encoded in the business object definition rather than on instructions hard-coded in the connector module.

Business object metadata includes the structure of a business object, the settings of its attribute properties, and the content of its application-specific information. Because connector modules are metadata driven, they can handle new or modified business objects without requiring modifications to the connector-module code.

The vision connector framework uses the value of the verb application-specific information in the top-level business object to call the appropriate connector module to process the business object. The verb application-specific information provides the classname of the connector module.

The verb application-specific information of most top-level business objects must identify the classname of the connector module. The syntax of this verb application-specific information is:

```
AppSpecificInfo = PartialPackageName.ClassName,
```

For example,

```
AppSpecificInfo = sap.sapextensionmodule.VSapBOHandler,
```

In this example, `sap.sapextensionmodule` is the partial package name, and `VSapBOHandler` is the classname. The full package name includes the `com.crossworlds.connectors` prefix, which the WebSphere business integration system adds to the name automatically. In other words, the full text of the example is:

```
com.crossworlds.connectors.sap.sapextensionmodule.VSapBOHandler
```

Note: The verb application-specific information of most top-level business objects must use a comma (**,**) delimiter after the connector classname. However, the Server verb, which is used by the RFC Server Module, is delimited instead by a semi-colon (**;**). For information about the Server verb, see "How the RFC Server Module works" on page 185 and "Supported verbs" on page 194.

You need not specify the package name and classname for the verb application-specific information if the business object is used:

- by the ALE module to process application events; however, when you use the ALE module to process service call requests, you must specify the package name and classname
- by the ABAP Extension module, which uses the default business object handler (`sap.sapextensionmodule.VSapBOHandler`)

Important: Customer-generated connector modules that process business objects for the BAPI and RFC Server modules must specify a full package name, which must begin with `bapi`. For example, `bapi.client.Bapi_customer_getdetails2`. The full package name in this example is `bapi.client`, and the classname is `Bapi_customer_getdetail2`.

Most business object processing is specific to each connector module. By default the connector uses the ABAP Extension Module. For more information on business object processing for the ABAP Extension Module, see “Business object processing” on page 36 and “Business object data routing to ABAP handlers” on page 61.

For more information on specifying verb application-specific information for the ALE module, see “Processing multiple IDocs with a wrapper business object” on page 156.

Processing multiple concurrent interactions

The system-wide Connector Framework can create separate threads for processing an application event and a business object request. When processing multiple requests from the integration broker, it can create multiple threads to handle multiple business object requests. For example, when InterChange System is the integration broker, the connector can receive multiple business object requests from multiple collaborations or from a multi-threaded collaboration.

Important: BAPI business object handlers generated before the connector for SAP version 4.3.0 are not thread-safe. To guarantee data consistency and integrity when using multi-threading, you must regenerate these business object handlers. The business objects do no require any change.

Figure 3 illustrates the multi-threading architecture.

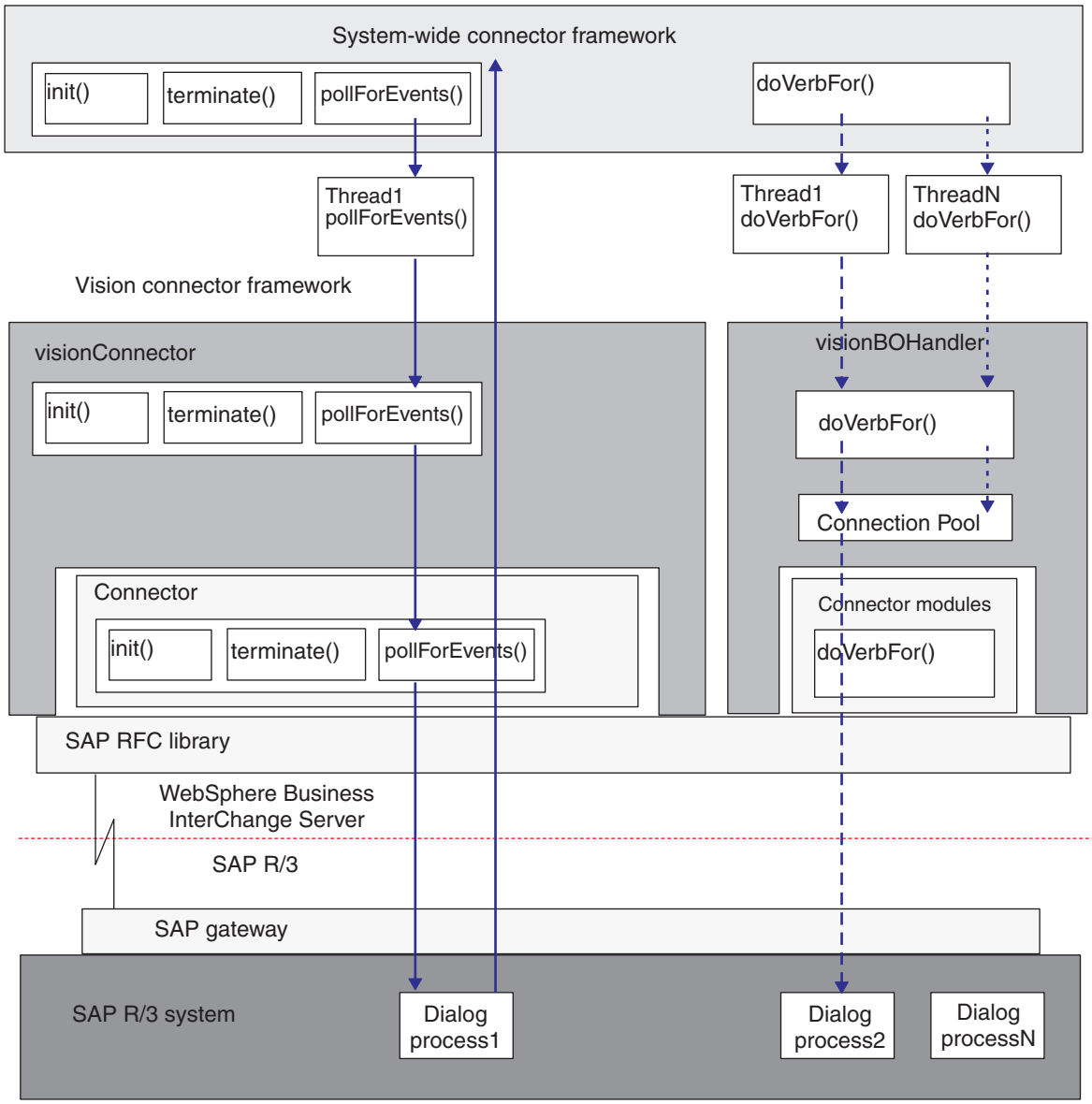


Figure 3. Multi-Threading Architecture of the Connector for SAP

Event processing

The connector performs the following steps when handling a poll call:

1. The system-wide Connector Framework creates a single dedicated thread to handle poll calls. This thread calls the `pollForEvents()` method of the vision connector framework at the frequency specified in the `PollFrequency` configuration property.
2. The thread polls SAP, which uses a dialog process to locate and return the event.

   Note: If the connector's MaxNumberOfConnections configuration property evaluates to a number greater than 1, the vision connector framework dedicates a connection to SAP for polling. If `MaxNumberOfConnections` evaluates to 1, event and service-call request processing share a single connection to SAP.

The polling thread dies only when the connector shuts down.

Note: Because the RFC Server connector agent pushes events out of SAP instead of polling for events, it spawns its own threads instead of using threads created by the connector framework. Because the ALE connector agent uses the RFC Server connector agent to access events, it also it spawns its own threads instead of using threads created by the connector framework when it processes events.

Request processing

Independently of polling, the system-wide Connector Framework can create multiple request-processing threads, one for each request business object. Each request thread instantiates the appropriate business object handler.

For example, when processing business object requests from InterChange Server, the number and type of business object handlers depends on the number and type of the collaborations sending the requests:

- If multiple collaborations send business objects, each request thread instantiates a business object handler of the appropriate type.
- If a multi-threaded collaboration sends multiple business objects of the same type, the request threads instantiate an equal number of business object handlers of that type.

If the connector's MaxNumberOfConnections configuration property evaluates to a number greater than 1, the vision connector framework dedicates one connection to SAP for polling and allocates the remaining connections to a pool used only for request processing.

As illustrated in Figure 3, the connector performs the following steps when handling a business object request:

1. The system-wide Connector Framework creates a separate thread for each business object request. Each thread calls the `doVerbFor()` method of the Vision business object handler.
2. If the connector's `MaxNumberOfConnections` configuration property evaluates to a number greater than 1, the Vision business object handler checks the vision connector framework's connection pool to determine if a connection handle is available.
   - If the handle is available, the thread sends the request to SAP, which uses a dialog process to handle the request.
   - If the handle is not available, the thread waits until one becomes available. Thread sequencing determines the order in which each business object handler thread claims or waits for an available connection handle.

   If the connector's `MaxNumberOfConnections` configuration property evaluates to 1, the Vision business object handler shares a connection with event processing.
3. SAP releases the dialog process after it completes processing and sends a return code.
4. The connector releases the connection handle after it receives the return code from SAP.

Setting the number of available connections

Use the "MaxNumberOfConnections" on page 22 configuration property to specify the maximum number of connection handles available. The number of connections cannot exceed the number of dialog processes.

SAP locks the dialog process while processing an interaction, releasing it only when the interaction completes. Therefore, multiple concurrent requests lock an equal number of dialog processes until processing finishes.

Important: Before setting a value for MaxNumberOfConnections, contact your SAP BASIS administrator to determine an appropriate value to maximize throughput without negatively affecting performance on the application server.

Setting multiple connections

By default the connector supports only single-threading. To cause the connector to support multiple threads, remove the following flag from your connector startup script:

UNIX

```
-tMAIN_SINGLE_THREADED
```

End of UNIX

Windows

```
-tSINGLE_THREADED
```

End of Windows

For more information, see the *Connector Development Guide for Java*.

Chapter 2. Installing and configuring the connector

This chapter describes the installation and configuration of the connector component of the Adapter Guide for mySAP.com (R/3 V.4.x). The chapter assumes that all the necessary files were installed when the WebSphere business integration system was installed.

This chapter contains the following sections:

Important: If you are upgrading versions of the connector, you must replace the connector Java Archive files (`.jar`). You also need to upgrade the connector transport files as well as any business object transports that you have previously installed. Depending on changes made to the connector, you may need to load a new copy of the `SAPConnector.txt` file into your repository. See the Release Notes for more information.

Compatibility

This section contains compatibility information for the IBM WebSphere Business Integration Adapter for mySAP.com.

The adapter runs on the following operating systems:

- Microsoft Windows 2000
- OS400 V5R2 (5722-SS1)
- Red Hat Enterprise Linux WS/ES/AS for Intel 2.1, 2.4 Kernel
- SuSE Linux Enterprise Server 7.3, 2.4 Kernel

mySAP.com compatibility

The adapter supports the following mySAP.com products:

- SAP R/3 V. 3.1I
- SAP R/3 3.X systems
- Other SAP solutions running on BAP with Application Server 6.2 (BAPI)

Integration compatibility

The adapter framework that an adapter uses must be compatible with the version of the integration broker (or brokers) with which the adapter is communicating. Version 5.4.x of the adapter for mySAP.com V4.x is supported on the following adapter framework and integration brokers:

Adapter framework: WebSphere Business Integration Adapter Framework versions 2.1, 2.2, 2.3.x, and 2.4

Integration broker: WebSphere InterChange Server, version 4.2.x

Prerequisites for installation

All of the components of the connector can be found in the `\connectors\SAP` directory.

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.

Prior to installing the connector component of the IBM WebSphere Business Integration Adapter for mySAP.com:

- Download the SAP Java API. SAP calls their Java API the Java Connector (SAPJCo).

  Download the SAPJCo for the operating system on which your connector is running. The SAPJCo is available for download from SAP's website at `http://service.sap.com/connectors`. You must have an SAPNet account to access the SAPJCo (if you do not already have one, contact your local SAP BASIS administrator).

  Add these files to the `\connectors\SAP` directory after you install the connector. For steps on installing the connector, see "Installing SAP's Java Connector (SAPJCo)" on page 15.
- The SAP Adapter currently supports SAPJCo V.2.0.9. If the SAPJCo version mentioned in this document is not available for download from SAP Service Marketplace, please contact your IBM representative.
- Install the SAP client on the same machine on which you are installing the connector.
- Install the most recent SAP Support Package for your version of SAP.

  SAP delivers Support Packages for: Basis, the R/3 application, ABAP, and HR. They provide bug fixes for the ABAP code in the SAP application. Use an updated SAP kernel. The kernel is the executables, written in C++, that carry out transports, interface with the operating system, communicate with the database, and run the system.
- Set up a CPIC user account in the SAP application. Give this account the necessary privileges to manipulate the data required by the business objects supported by the connector.

  For example, if the connector must perform certain SAP business transactions, the connector's account in the SAP application must have the permissions to perform these transactions. You must set the connector-specific configuration properties `ApplicationUserName` and `ApplicationPassword` using this account information. For more information on how to set these properties, see "Configuring the connector" on page 17.
- Set up a user account in SAP with privileges to install and administer the connector. The account should have the following characteristics:
  - A valid SAP user name and password
  - ABAP developer access
  - Table configuration access
  - Administration access for transactions SM21 and SM50 to administer and monitor the connector

- If using the ALE Module, see "Prerequisites to running the ALE Module" on page 131 for additional information on installing MQSeries queues.

Installing the connector component

After your WebSphere business integration system has been installed, you can install additional adapters from the product CD at any time. To do this, insert the product CD, run the installation program, and choose the adapters that you want to install.

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

This section describes only the installation of the connector's application-specific component. By default, the connector uses the ABAP Extension Module, so you must install the transport files that support that connector module. After you have installed and configured the connector, you must install the ABAP Extension Module. For more information on installing the ABAP Extension Module, see Chapter 4, "Installing and customizing the ABAP Extension module," on page 47.

The IBM WebSphere Business Integration Adapter for mySAP.com connector can be installed on a UNIX or Windows machine. The connector consists of three parts that need to be installed: the connector's application-specific component, SAP's RFC library, and any SAP transport files delivered with the product and required to support the connector.

After you have installed the required connector files, you must download and install the Java Connector (SAPJCo) files. For more information on downloading the SAPJCo files, see "Prerequisites for installation" on page 12. For more information on installing the SAPJCo files, see "Installing SAP's Java Connector (SAPJCo)" on page 15.

The connector files must be installed on a machine that is capable of acting as an SAP client. By default, the connector JAR files are installed with the integration broker.

Installing on a UNIX system

To install the connector on a UNIX system, run the Installer for IBM WebSphere business integration adapter, and select mySAP.com IBM WebSphere Business Integration Adapter for mySAP.com. Table 1 lists the files used by the connector that runs in a UNIX environment.

Table 1. WBIA: UNIX file

Directory/Filename	Description
`connectors/SAP/bapi/client`	Directory containing the BAPI Module business object handler files
`connectors/SAP/bapi/server`	Directory containing the RFC Server Module business object handler files
`connectors/SAP/dependencies`	Directory containing all version-specific transport files
`connectors/SAPmessages`	Directory containing the `SAPConnector.txt` file
`connectors/SAP/samples`	Directory containing sample ABAP objects
`connectors/SAP/utilities`	Directory containing the `generatedfiles` subdirectory, into which you can put files generated by SAPODA
`connectors/SAP/CWSAP.jar`	Connector class files

Table 1. WBIA: UNIX file (continued)

Directory/Filename	Description
connectors/SAP/start_SAP.sh	System startup script for the connector. This script is called from the generic connector manager script. The product installer creates a customized wrapper for this connector manager script. When the connector works with WebSphere InterChange Server, use this customized wrapper to start and stop the connector.
repository/SAP	Directory containing the CN_SAPSAP.txt file
/lib	Contains the WBIA.jar file
/bin	Contains the CWConnEnv.sh file

Before you can use the connector, you must configure the connector from the installer's Connector Configuration screen. From this screen:

- Choose SAP from the Select Connector Name list.
- Click Install to have Installer generate the customized SAP wrapper, connector_manager_SAP.

Note: For more information on installing the connector component, refer to the *System Installation Guide for UNIX*.

Installing on a Windows system

To install the connector on a Windows system, run Installer for IBM WebSphere Business Integration Adapter, and select the IBM WebSphere Business Integration Adapter for mySAP.com. Installer installs standard files associated with the connector. Table 2 lists the standard files installed in a Windows environment.

Table 2. WebSphere Business Integration Adaptor: Windows file

Directory/filename	Description
connectors\SAP\bapi\client	Directory containing the BAPI Module business object handler files
connectors\SAP\bapi\server	Directory containing the RFC Server Module business object handler files
connectors\SAP\dependencies	Directory containing all version-specific transport files
connectors\SAPmessages	Directory containing the SAPConnector.txt file
connectors\SAP\samples	Directory containing sample ABAP objects
connectors\SAP\CWSAP.jar	Connector class file
connectors\SAP\start_SAP.bat	Batch file used to start the connector
repository\SAP	Directory containing the CN_SAPSAP.txt file
\lib	Contains the WBIA.jar file
\bin	Contains the CWConnEnv.bat file

Installer adds a menu option for the connector's application-specific component to the IBM WebSphere business integration adapters menu. For a fast way to start the connector, create a shortcut to this component on the desktop.

Installing on an iSeries system

To install the connector on an iSeries system, run Installer for IBM WebSphere Business Integration Adapter, and select the IBM WebSphere Business Integration

Adapter for mySAP.com. Installer installs standard files associated with the connector. Table 3 lists the standard files installed in an iSeries environment.

Table 3. WebSphere Business Integration Adaptor: iSeries file

Directory/filename	Description
connectors\SAP\bapi\client	Directory containing the BAPI Module business object handler files
connectors\SAP\bapi\server	Directory containing the RFC Server Module business object handler files
connectors\SAP\dependencies	Directory containing all version-specific transport files
connectors\SAPmessages	Directory containing the SAPConnector.txt file
connectors\SAP\samples	Directory containing sample ABAP objects
connectors\SAP\CWSAP.jar	Connector class file
connectors\SAP\start_SAP.sh	Batch file used to start the connector
repository\SAP	Directory containing the CN_SAPSAP.txt file
\lib	Contains the WBIA.jar file
\bin	Contains the CWConnEnv.sh file

Installer adds a menu option for the connector's application-specific component to the IBM WebSphere business integration adapters menu. For a fast way to start the connector, create a shortcut to this component on the desktop.

Installing SAP's Java Connector (SAPJCo)

The *IBM WebSphere Business Integration Adapter for mySAP.com* integration broker requires the use of SAP's Java Connector (SAPJCo). If you have already followed instructions for installing the connector component, you should have already downloaded SAP's Java Connector (SAPJCo) as described in "Prerequisites for installation" on page 12. If you have not downloaded the SAPJCo, download and unzip it now.

After you have installed the files delivered with SAPODA, copy the following unzipped SAPJCo files into your environment.

Unix

From the zipped file, extract the executable jar file (.jar extension) and the runtime libraries (.o for AIX, .so for Solaris, and .a for HP-UX).

Windows

From the zip file, extract the executable jar file, (`.jar` extension) and the runtime libraries (`.dll` extension). If you have already followed instructions for installing the IBM WebSphere Business Integration Adapter for mySAP.com on the same machine on which you install SAPODA, copy these files from the `\connectors\SAP` directory to the `\ODA\SAP` directory. If you install SAPODA on a different machine from the connector, after you unzip the SAPJCo files, copy these four files to the `\ODA\SAP` directory. For Windows, the `librfc32.dll` requires one or more C runtime dlls. The C runtime dlls depend on the version of the SAP release being used. Through SAP release 45B, the C runtime dll required is `msvcrt.dll` version 5.00.7022 or newer. Starting with SAP release 46A, the C runtime dlls required are `msvcrt.dll` version 6.00.8267.0 or newer and `msvcp60.dll` version 6.00.8168.0 or newer. The dll or dlls should be copied into the C:\WINNT\system32 directory. This dll or these dlls may already be present and if not, can be found on the "Presentation CD" that contains the Windows SAPGUI setup in the folder *<cddrive>*`:\GUI\Windows\Win32\system`. See SAP OSS note number 0182805 for more information.

Installing connectors on remote machines

You can install and run the connector on a remote machine. Install the integration broker on one machine and the connector on another machine. It is recommended but not required that both machines be on the same subnet.

Installing multiple connectors

To enable the integration broker to handle multiple business objects for SAP at the same time, you may want to install and configure multiple connector components for an SAP system and customize each connector to handle specific business objects.

Each connector component can subscribe to certain business objects depending on their type (such as Customer or Purchase Order). Because you can have multiple connectors accessing the same SAP application, each connector can process events and pass them on to the integration broker. In addition, multiple connectors can support multiple business object requests at the same time. This increases throughput and speeds up the transfer of data into and out of the SAP application.

It is recommended that you choose a unique naming convention for each connector component. For example, if you are using two connectors you could name them `SAP1Connector` and `SAP2Connector`.

To install and set up multiple connector components, do the following:

1. Install each of the connectors as described in this chapter. This includes the connector shared library files. Give a unique name to each connector you install, and verify that you have the supporting connector files.

   If you are installing multiple connectors on the same machine, you need only make a copy of the shared library files and rename them. You do not need to install the transports again.
2. Create a copy of the startup script:

- On UNIX, make a copy of the existing `connector_manager_SAP` file for starting the connector, and rename the file to match the name of the connector.
- On Windows, make a copy of the existing shortcut to the `start_SAP.bat` file, and rename the shortcut file to match the name of the connector. Add the name of the connector as a parameter of the connector shortcut.

3. Make a copy of the connector definition file (`CN_ConnectorName`), rename it to match the new connector name, and then load it into the IBM WebSphere repository (if the IBM WebSphere InterChange Server is the integration broker).
4. Make a copy of the connector class file, `CWSAP.jar`and rename it to the unique connector name, such as `CWSAP1.jar`.
5. Initialize the connector configuration properties so that all connectors poll the same SAP application for events.
6. Only if the the IBM WebSphere InterChange Server is the integration broker, add map references for each connector.
7. Specify the business objects supported by each connector.
8. Only if WebSphere InterChange Server is the integration broker, assign collaborations to the appropriate connectors. Currently, a collaboration can be handled by only one connector. If collaborations are already set up, you may need to stop them and then rebind the ports.
9. If you are using the ABAP Extension Module for business object handing, set up the distribution of events to each connector that you install. Use IBM CrossWorlds Station (transaction `/n/CWLD/HOME`). See "Setting up event distribution" on page 53 for instructions on setting up event distribution for each combination of business object, integration broker, and connector.

Important: If a business object is not configured to go to a particular connector, the business object is sent to the next connector that polls for events. If a business object is configured to go to a particular connector, as for example during the testing phase, but the connector is not used in the production phase, the event queue for the connector fills up. To remedy this situation, delete the connector/business object configuration in the Event Distribution window (transaction /CWLD/RH).

Configuring the connector

You must configure the connector's standard and connector-specific connector configuration properties before you can run it. To configure connector properties, use Connector Designer. Access this tool from the System Manager.

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

Standard configuration properties

Standard configuration properties provide information that all connectors use. See Appendix B, "Standard configuration properties for connectors," on page 241 for documentation of these properties.

Table 4 on page 18 provides information specific to this connector about configuration properties in the appendix.

Table 4. Property information specific to this connector

Property	Note
CharacterEncoding	The connector does not use this property.
Locale	Because this connector has been internationalized, you can change the value of this property. See release notes for the adapter to determine currently supported locales.
PollFrequency	If using the RFC Server Module or the ALE Module for event processing, do not set this property's value to key or to no. Setting the value to key or no prevents the connector from instantiating these modules at startup.

You must provide a value for the ApplicationName configuration property before running the connector.

Connector-specific configuration properties

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

Table 5 is a quick reference for the connector-specific configuration properties. The modules column contains a list of the connector modules that use the associated property.

Table 5. Quick reference for connector-specific configuration properties

Name	Possible values	Default value	Modules
ABAPDebug	true or false	false	ABAP Extension BAPI HDR
AleEventDir	*path*		ALE
AleUpdateStatus	true or false	false	ALE
AleSelectiveUpdate	*IDocType:MessageType*		ALE
AleStatusMsgCode	*MessageCode*		ALE
AleSuccessCode	52 or 53	52	ALE
AleFailureCode	68 or 58	68	ALE
AleSuccessText	*SuccessText*		ALE
AleFailureText	*FailureText*		ALE
ApplicationPassword		SOFTWARE	All
ApplicationUserName		CROSSWORLDS	All
ArchiveDays			ALE
Client			All
Group	*any valid name of the logon group that represents a group of application servers*		All
gwService	*Gateway server identifier*	sapgw00	RFC Server ALE
Hostname	IP-address or *server-name*		All
InDoubtEvents	Reprocess, FailOnStartUp, LogError or Ignore	Ignore	ABAP Extension
Language		E	All
MaxNumberOfConnections		2	ABAP Extension, ALE, BAPI HDR
Modules	*ModuleName*		All
Namespace	true or false	true	ABAP Extension

Table 5. Quick reference for connector-specific configuration properties (continued)

Name	Possible values	Default value	Modules
NumberOfListeners	*any positive integer*	1	RFC Server, ALE
PollQuantity	*any positive integer*	20	ABAP Extension, ALE
RefreshLogonCycle	true	true	All
RfcProgramId	*program ID*	CWLDSERVER	RFC Server , ALE
RfcTraceOn	true or false	false	All
SAPALE_Archive_Queue	*any valid MQ Series queue name*		ALE
SAPALE_Event_Queue	*any valid MQ Series queue name*		ALE
SAPALE_Wip_Queue	*any valid MQ Series queue name*		ALE
SAPALE_Error_Queue			
SAPALE_Unsubscribed_Queue			
SAPSystemID	*logical name of the SAP R/3 System*		All
SAPtid_MQChannel	*any valid MQ channel*		ALE
SAPtid_MQPort	*any valid MQ port*		ALE
SAPtid_Queue	*any valid MQ queue name*		ALE
SAPtid_QueueManager	*any valid MQ queue manager name*		ALE
SAPtid_QueueManagerHost	*any valid MQ queue manager host name*		ALE
SAPtid_QueueManagerLogin	*any valid MQ queue manager login name*		ALE
SAPtid_QueueManagerPassword	*any valid MQ queue manager password*		ALE
Sysnr	*system-number*	00	BAPI, RFC Server
DateTimeFormat	*nothing or legacy*		All
TransIdCollabName			**No longer supported**
UseDefaults	true or false	false	ABAP Extension ALE BAPI

ABAPDebug

Specifies whether the connector invokes the ABAP Debugger for the appropriate function module when the connector begins processing a business object. When this property is set to true, the connector opens the ABAP Debugger for the following connector modules:

- ABAP Extension—when processing events out of SAP and service call requests into SAP
- BAPI—only when processing service call requests into SAP
- Hierarchical Dynamic Retrieve—when processing service call requests into SAP

The connector invokes the ABAP Debugger only if you have:

- Changed the default value of the **"ApplicationUserName" on page 22** configuration property from CROSSWORLDS to a Dialog user with proper user authorizations.
- Set the ABAPDebug property to true.

Note: You can add breakpoints only after the debugger opens.

Important: This property should always be set to `false` in a production environment.

The default value is `false`.

AleEventDir

Specifies the location of the root directory (`\ale`) for the `event` directory used by the ALE Module to log and recover events. When the connector starts for the first time, if it does not find the root directory in the directory from which the connector is started, it creates it and the `event` subdirectory:

- If the path is specified in this property, it uses that path to create the directory.
- If no path is specified, it creates the root directory in the directory from which the connector is started.

For example, if your connector is located in `\connectors\SapConnector1` (within the product directory), the connector creates the following directory:

```
\connectors\SapConnector1\ale
```

UNIX

If you are not in the connector's directory when you start the connector for the first time, the connector creates the root directory in the directory from which you start the connector regardless of the value of this property.

For more information, see Chapter 11, "Overview of the ALE module," on page 125.

The default value is:

UNIX

$<ProductNameDir>/connectors/SAP/ale

Windows

%ProductNameDir%\connectors\SAP\ale

AleUpdateStatus

Specifies whether an audit trail is required for all message types. This property must be set to `true` to cause the connector to update a standard SAP status code after the ALE Module has retrieved an IDoc object for event processing.

For more information, see Chapter 11, "Overview of the ALE module," on page 125.

The default value is `false`.

AleSelectiveUpdate

Specifies which IDocType and MessageType combinations are to be updated when the connector is configured to update a standard SAP status code. You can define values for this property only if `AleUpdateStatus` has been set to `true`.

The syntax for this property is:

IDocType:MessageType[,*IDocType:MessageType* [,...]]

where a colon (:) delimiter separates each IDocType and MessageType, and a comma (,) delimiter separates entries in a set. The example below illustrates two sets. In the example, MATMAS03 and DEBMAS03 are the IDocs, and MATMAS and DEBMAS are the message types:

```
MATMAS03:MATMAS,DEBMAS03:DEBMAS
```

For more information, see Chapter 11, "Overview of the ALE module," on page 125.

AleStatusMsgCode

If required, specifies the message code to use when the connector posts the ALEAUD Message IDoc (ALEAUD01). Configure this message code in the receiving Partner Profile. You can set a value for this property only if AleUpdateStatus has been set to true.

For more information, see "Configuring SAP To update IDoc status" on page 133.

AleSuccessCode

Specifies the success status code for Application Document Posted. You must specify a value for this property (52 or 53) to cause the connector to update the SAP success status code after the ALE Module has retrieved an IDoc object for event processing. SAP converts this value to status 41 (Application Document Created in Receiving System).

For more information, see Chapter 11, "Overview of the ALE module," on page 125.

AleFailureCode

Specifies the status code for dispatch failure. You must specify a value for this property (68 or 58) to cause the connector to update the SAP failure status code after the ALE Module has retrieved an IDoc object for event processing. SAP converts this value to 40.

For more information, see Chapter 11, "Overview of the ALE module," on page 125.

AleSuccessText

Specifies the descriptive text for successful Application Document Posted. Specifying a value for this property is optional, even when you set AleUpdateStatus to true.

For more information, see Chapter 11, "Overview of the ALE module," on page 125.

AleFailureText

Specifies the descriptive text for dispatch failure. Specifying a value for this property is optional, even when you set AleUpdateStatus to true.

For more information, see Chapter 11, "Overview of the ALE module," on page 125.

ApplicationPassword
Password for the connector's user account on the SAP application. The default is `SOFTWARE`.

ApplicationUserName
Name of the connector's user account on the SAP application. The default is `CROSSWORLDS`.

ArchiveDays
The ArchiveDays connector configuration property determines the number of days after which TIDManagement files should be deleted from the request directory. The default value maintained internally is seven days.

Client
Client number under which the connector logs in, often `100`.

Group
When configuring the connector for load balancing, specifies the name of the logon group that represents a group of application servers. For more information, see "Taking advantage of load balancing" on page 26.

gwService
Gateway server identifier; often `sapgw00`. The `00` is the system number of the server running the SAP Gateway (usually an application server) and may not be `00` if you have more than one. The default is `sapgw00`.

Hostname
When configuring the connector for load balancing, specifies the name of the message server. When configuring the connector to run without load balancing, specifies the IP address or the name of the application server that the connector logs in to. In both cases, the connector assumes that the name of the gateway host is the same as the value specified for this property.

InDoubtEvents
InDoubtEvents describes how to handle in-progress events in the events table. `Reprocess` reprocesses the in-progress events in the events table. `FailOnStartup` will shut down the connector and log a fatal error when in-progress events are found. `LogError` logs an error notifying that in-progress events are in the event table. `Ignore`, ignore the in-progress events.

Language
Language in which the connector logs in. The default is `E`, for English.

MaxNumberOfConnections
The maximum number of concurrent interactions possible between the connector and the SAP application. These interactions include polling for events and handling service call requests. Only the ABAP Extension, BAPI, and ALE Modules use this property. The ALE Module uses this property only for service call requests.

Because each interaction uses a dialog process on the SAP application server, the number of connections cannot exceed the number of dialog processes available. For more information, see "Processing multiple concurrent interactions" in Chapter 1.

If no value is specified for this property, the connector uses the default value of 2.

Modules
Identifies the module used by the connector to carry out the `init()`, `pollForEvents()`, and `Terminate()` requests. Specifically, it specifies the connector module used by the Vision Connector framework. Specify multiple connector modules by separating each value with a comma. Do not add spaces.

The supported connector modules and the syntax to specify them is as follows:

ABAP Extension Module—`Extension`

ALE Module—`ALE`

BAPI Module—`Bapi`

RFC Server Module—`RfcServer`

Namespace
Specifies whether or not the connector uses the ABAP components defined in the connector's namespace `/CWLD/`. The value must be set to `true` in order for the connector to use the ABAP components defined in the namespace. The default is `true`.

NumberOfListeners
Specifies the number of listener threads that are created when the connector is initialized. A listener thread can handle one request at a time. Each listener thread handles a single event at a time; therefore, if you have multiple listener threads, the connector can handle multiple events concurrently. The default is 1.

It is recommended that you have no more listener threads than the available work processes in SAP.

PollQuantity
Defines the maximum number of events picked up for a single poll. The default is `20`.

RefreshLogonCycle
Specifies whether all resources are to ber freed for an SAP client connection. The default is `false`.

RfcProgramId
Identification that the connector registers in the SAP Gateway so that the listener threads can process events from RFC-enabled functions. This value must match the Program ID registered in the SAP application (transaction SM59). The default is `CWLDSERVER`.

For more information on configuring the Program ID in the SAP application, see "Registering the RFC Server Module with the SAP gateway" on page 189.

RfcTraceOn
Specifies whether or not to generate a text file detailing the RFC activity for each listener thread. You can specify a value of `true` or `false`. A value of `true` activates tracing, which generates a text file. It is recommended that you use these text files in a development environment only, because the files can grow rapidly. The default is `false`.

SAPALE_Archive_Queue
Specifies the MQ Series queue that archives TIDs and IDoc data after the ALE Module has finished processing events. For more information, see Chapter 11, “Overview of the ALE module,” on page 125.

There is no default value.

SAPALE_Event_Queue
Specifies the MQ Series queue that stores TIDs and IDoc data during the ALE Module’s processing of events. For more information, see Chapter 11, “Overview of the ALE module,” on page 125.

There is no default value.

SAPALE_Wip_Queue
Specifies the MQ Series work-in-progress (wip) queue that holds TIDs and IDoc data while the ALE Module builds the MQ message for the event queue. After the connector receives all data for an event, it moves the data in this queue to the `SAPALE_Event_Queue`. For more information, see Chapter 11, “Overview of the ALE module,” on page 125.

There is no default value.

SAPALE_Error_Queue
Defines a queue to handle MQ messages that fail between the WIP Queue and the Event Queue. For more information, see Chapter 11, “Overview of the ALE module,” on page 125.

SAPALE_Unsubscribed_Queue
Defines a queue to collect unsubscribed IDoc objects. Unsubscribed IDoc objects previously were placed in the Archive queue. These messages can be resubmitted using the event management utility. The connector now checks for subscriptions when processing the data from SAP to the connector, resulting in transactions remaining in SAP until the collaboration is started. For more information, see Chapter 11, “Overview of the ALE module,” on page 125.

SAPSystemID
When configuring the connector for load balancing, specifies the logical name of the SAP R/3 System, which is also known as `R3name`. For more information, see “Taking advantage of load balancing” on page 26.

SAPtid_MQChannel
Specifies the Client channel for the MQ Series queue manager. For more information, see Chapter 11, “Overview of the ALE module,” on page 125.

There is no default value.

SAPtid_MQPort
Specifies the port used to communicate with the MQ Series queue manager that handles the queues for the ALE Module. For more information, see Chapter 11, “Overview of the ALE module,” on page 125.

There is no default value.

SAPtid_Queue
Specifies the MQ Series queue on which messages containing the TID and TID status reside. This property is used by the ALE Module only when processing requests. For more information, see Chapter 11, “Overview of the ALE module,” on page 125.

There is no default value.

SAPtid_QueueManager
Name of the MQ Series queue manager for the queues that store TIDs and IDoc data. This property is used by the ALE Module to process events and requests. For more information, see Chapter 11, “Overview of the ALE module,” on page 125.

There is no default value.

SAPtid_QueueManagerHost
Name of the host where the MQ Series queue manager resides. This property is used by the ALE Module to process events and requests. For more information, see Chapter 11, “Overview of the ALE module,” on page 125.

There is no default value.

SAPtid_QueueManagerLogin
User name to log into the MQ Series queue manager. This property is used by the ALE Module to process events and requests. For more information, see Chapter 11, “Overview of the ALE module,” on page 125.

There is no default value.

SAPtid_QueueManagerPassword
Password for the user who logs into the MQ Series queue manager. This property is used by the ALE Module to process events and requests. For more information, see Chapter 11, “Overview of the ALE module,” on page 125.

There is no default value.

Sysnr
System number of the application server. The value is a two-digit number, often `00`. The default is `00`.

DateTimeFormat
Preserves the delimiters provided with `DATE` and `TIME` field values. If set to `Legacy`, the connector will preserve the delimiters for `DATE` and `TIME` fields. Otherwise, the delimiters will be removed and the value’s length will conform to the attribute defined length.

TransIdCollabName

Important: The connector no longer supports this property.

UseDefaults
On a Create or Update operation, if UseDefaults is set to `true`, the Adapter Framework for the integration broker, checks whether a valid value or a default value is provided for each business object attribute marked as required. If a value is provided, the Create or Update operation succeeds. If the parameter is set to `false`, the connector checks only for a valid value and causes the Create or Update operation to fail if it is not provided. The default is `false`.

Connector startup

For information on starting a connector, stopping a connector, and the connector's temporary startup log file, see *Implementation Guide for WebSphere InterChange Server*.

Taking advantage of load balancing

Load balancing at logon increases the efficiency of defined workgroups by:

- Improving performance
- Decreasing consumption of system resources
- Distributing users across available application servers based on requirements for workgroup service and load sensitivity

Starting the connector with the load balancing feature initiates communication with the message server specified by the Hostname property. The message server then finds the application server with the least load. Once this application server is determined, the message server routes all future RFC communication with the connector through this one application server. The connector is considered as one dialog user with the message server.

The load balancing feature works best in an SAP environment where the connector processes low volumes and the number of users is large. For high volumes consider connecting directly to one of your larger application servers.

For information on configuring the connector for load balancing, see the description of the following connector properties:

- "ApplicationPassword" on page 22
- "ApplicationUserName" on page 22
- "Client" on page 22
- "Group" on page 22
- "Hostname" on page 22
- "InDoubtEvents" on page 22
- "SAPSystemID" on page 24

Starting multiple connectors

To start multiple connectors, follow these instructions:

1. Copy the connector definition, giving the new definition an appropriate name. For example, name the new definition `SAP2Connector`.
   - If the connector uses WebSphere MQ Integration broker as the integration broker, use a system command to copy the definition file (by default, `\repository\Sap\CN_SAP.txt`) to `CN_SAP2.txt`. Then open the definition file with a text editor and replace `SAPConnector`with `SAP2Connector`for "name" and the attribute value of the Application name Property.
   - If the connector uses WebSphere Integration broker as the integration broker, use System Manager to copy and name the definition.
2. Make a copy of the entire `\connectors\SAP` directory, and change the name to match the new connector name, for example `\connectors\SAP2`.
3. In the new connector directory, make a copy of the `CWSAP.jar` file, and then rename it to match the name of the new connector, for example `CWSAP2.jar`.

4. For Windows, create a connector shortcut and point it to the new connector startup file, `start_SAP2.bat`, in the `\connectors\SAP2`directory, for example.
5. Copy the `SAPConnector.txt` message file in the `\connectors\messages\` directory, and rename it to match the new connector name, for example `SAP2Connector.txt`

Repeat these steps for each connector.

Upgrading the connector

This section describes how to upgrade the connector:

- "Upgrading the connector for the ALE Module's management of TIDs"
- "Upgrading to the java-based connector" on page 28

Upgrading the connector for the ALE Module's management of TIDs

The ALE Module persistently stores IDoc objects and Transaction IDs (TIDs) for every transaction it receives from the SAP application. In releases of the connector prior to version 4.8.x, the connector used IBM WebSphere collaborations, business objects, and maps to store the data in the repository. Version 4.8.x of the connector replaces the previous way of TID management with the use of MQSeries queues.

Attention: To enable the ALE Module to process IDocs to and from the SAP application, you must upgrade the connector. However, it is imperative that you allow the current IDoc processing cycle to complete before upgrading the connector.

Before you upgrade the connector to enable the ALE Module to process IDocs, you must complete the processing of current files in the event and WIP directories. Also, check the archive directory for failed and unsubscribed events.

To complete the processing of current files in the event and WIP directories:

- Temporarily halt the transmission of IDocs both to and from the connector.
- Verify that there are no IDocs (files) in the following directories when you upgrade:

UNIX

```
$CROSSWORLDS/connectors/SAP/ale/events
$CROSSWORLDS/connectors/SAP/ale/wip
```

Windows

```
%CROSSWORLDS%\connectors\SAP\ale\events
%CROSSWORLDS%\connectors\SAP\ale\wip
```

To complete processing of any failed and unsubscribed events:

- Temporarily halt the transmission of IDocs both to and from the connector.
- Verify the status of the IDocs (files) in the following directory when you upgrade:

UNIX

```
$CROSSWORLDS/connectors/SAP/ale/archive
```

Windows

```
%CROSSWORLDS%\connectors\SAP\ale\archive
```

- Correct any errors to the failed or unsubscribed events.
- Move the corrected file to the event directory for processing.

Note: If, when using transaction SM58, you find unsuccessfully processed IDocs in the SAP system, wait until the connector is upgraded to resubmit these IDocs. After you complete the upgrade of the connector, correct the errors and resubmit the IDocs for processing through the MQSeries queues using the new TID management.

Once these directories are clear, apply the upgrade and follow the configuration instructions in:

- "Connector-specific configuration properties" on page 18
- Chapter 11, "Overview of the ALE module," on page 125.
- Chapter 12, "Configuring the ALE module," on page 131

Upgrading to the java-based connector

The *IBM WebSphere Business Integration Adapter for mySAP.com* is delivered as a Java-based connector (connector version 4.0.0 and later). In previous releases, the connector was written in C++. The directory structure is `\connectors\SAP`.

The following procedure is for upgrading a C++ version of the connector to the Java version of the connector.

1. Rename the current connector directory.

   For example, `\connectors\SAP` becomes `connectors\SAP.old`.
2. Rename the connector message file.

   For example, `\connectors\messages\SAPconnector.txt` becomes `\connectors\messages\SAPconnector.txt.old`.
3. Copy the new connector directory and files to the `\connectors` directory.
4. Copy the new connector message file to the `\connectors\messages` directory.

Windows

Modify the connector shortcut to point to the `start_SAP.bat` file in the connector directory. For example, if you are using the connector that supports SAP R/3 version 4.x, modify the shortcut to point to the connector startup file `\connectors\SAP\start_SAP.bat`.

Comprehensive install and uninstall information

The information in this section describes how to install WebSphere Business Integration Adapters (WBIA).

Install the WBIA product by running a platform-specific executable for the installer. Table 6 lists the installer executable for each operating system. The installer executables are located in the `WebSphereBI` directory on the product CD.

Table 6. Platform-specific executables for WBIA Installer

Operating system	WBIA Installer executable file
Windows	`setupwin32.exe`
Linux	`setupLinux.bin`
iSeries	`setupiSeries.bin`

Note: These procedures assume that you are installing from a product CD. If you obtain your software from Passport Advantage, make sure you have downloaded it. Refer to your Passport Advantage information for those downloading instructions.

Note: If you are installing the adapters to communicate with InterChange Server, you must install the broker first. See the installation guide for InterChange Server on the appropriate platform for information on how to install the broker.

Important: Make sure you are logged in as the WebSphere business integration system administrator before you install the adapters. When you install on a UNIX computer, the permissions of the folders and files that are created are set based on the permissions of the user account that performs the installation.

Important: You must not install WBIA as root. The entry that is added to the Object Data Manager (ODM) when installing as root prevents you from using SMIT to uninstall other applications, so you should not install WBIA as root.

Part 2. ABAP Extension module

Chapter 3. Overview of the ABAP Extension module

This chapter describes the ABAP Extension Module of the IBM WebSphere Business Integration Adapter for mySAP.com (SAP R/3 Version 4.x). The ABAP Extension Module enables an integration broker to send business objects to and receive events from SAP R/3 application versions 4.0, 4.5, and 4.6.

This chapter contains the following sections:

ABAP Extension Module components

The ABAP Extension Module consists of components written in Java and ABAP. The Java components consist of the connector module and the SAP RFC libraries. SAP delivers their RFC libraries in Java and C. The ABAP components consists of various SAP application function modules, database tables, and programs. Some of these ABAP components are developed and delivered as part of the adapter and some are native to every SAP installation.

Figure 4 on page 34 illustrates the overall architecture of the ABAP Extension Module.

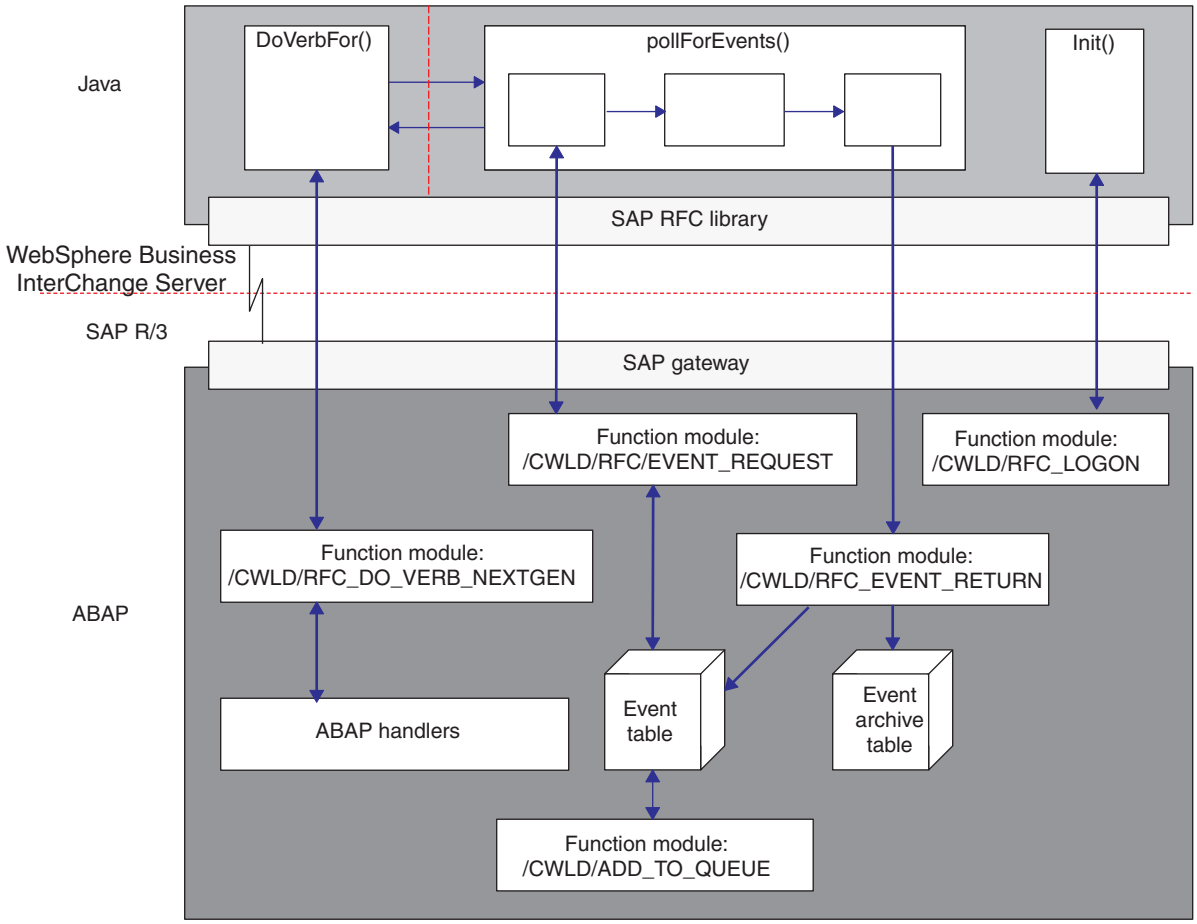


Figure 4. ABAP Extension Module architecture

Java components

The connector is delivered and run as a Java Archive (JAR) file. It handles the event delivery and event business object request processes. The SAP RFC library is delivered and run as a JAR file as well. It enables external programs to execute ABAP function modules within an SAP application.

The Java components:

- Open an RFC connection to the SAP application using the SAP RFC library and the SAP Gateway.
- Handle requests from the integration broker and pass the requests to an ABAP component of the connector.
- Poll the SAP application for events.

ABAP components

The ABAP components of the connector are function modules, programs, and database tables. These elements handle the event delivery and business object request processes initiated by the Java component. The ABAP components are delivered in connector transport files to be loaded into an SAP application; once loaded, they run as ABAP repository objects.

The ABAP components:

- Handle business object requests from the Java component by calling the appropriate function modules designed to handle a particular business object type and verb.
- Detect, trigger, and store events in the event table.
- Handle event requests and their subsequent return (event status update) from the Java component.

How the ABAP Extension Module works

Most of the functionality provided by the ABAP Extension Module occurs inside of the SAP application. For most of the virtual functions that every connector must implement, there is a corresponding ABAP function module in the SAP application. However, SAP does not provide ABAP function modules that support the specific requirements of the `init()`, `doVerbFor()`, and `pollForEvents()` methods, so these function modules have been developed and delivered as part of the connector module. While the Java components provide some functionality, the majority of the processing for these methods is done by the ABAP components in the SAP application.

Table 7 shows the virtual Java methods that the connector module implements and their corresponding ABAP components. Keep in mind that this is not a complete list of the ABAP components used by the connector.

Table 7. Java components and their corresponding ABAP components

Java components	ABAP components
`doVerbFor()`	`/CWLD/RFC_DO_VERB_NEXTGEN`
`getVersion()`	No implementation required
`getBOHandlerForBO`	No implementation required
`init()`	`/CWLD/RFC_LOGON`
`pollForEvents()`	`/CWLD/RFC_EVENT_REQUEST` `/CWLD/RFC_EVENT_RETURN`
`terminate()`	No implementation required

Together, these ABAP function modules are the core of the ABAP Extension Module. The following sections describe connector initialization, business object processing, and how the connector handles event notification.

The implemented functions are discussed in the rest of this chapter.

Initialization

The `init()` method calls the ABAP function module `/CWLD/RFC_LOGON` to validate that the destination SAP application is running and that the RFC library can be used to execute ABAP function modules. The `/CWLD/RFC_LOGON` function module is also called to process all of the in-progress events. All events in the event table that are marked with a status of event retrieved (status marked as R in the event table) will be processed based on the InDoubtEvents Connector Property. The default property value is Ignore. When event distribution is being used only the events belonging to that particular connector and server with a status of 'R' will be handled according to the connector property. If event distribution is not being used then all events with a status of 'R' will be handled according to the connector property. If the connector property equals reprocess, these events will be changed to a status of queued (marked as Q in the event table). When the connector polls for events, all events of 'Q' status will be processed, using `/CWLD/RFC_E`

VENT_REQUEST function module. If the connector property is equal to FailOnStartUp, a fatal error is logged within the SAP log and the local log file and the connector will shut down. An email is also sent notifying the user a fatal error has occurred. If the connector property is equal to LogError an error is logged within the SAP log and the local log file. The in-progress events are not processed and the connector does not shut down. If the connector property is equal to Ignore, the in-progress events are ignored and the connector polls as if there weren't any in-progress events in the event table.

If the function module does not execute successfully, the connector terminates.

Business object processing

All service call requests for SAP are initiated by the doVerbFor() method in the Java component of the connector module. The connector's ABAP function module /CWLD/RFC_DO_VERB_NEXTGEN and an ABAP handler in the ABAP component of the connector module handle the requests.

Figure 5 illustrates business object processing.

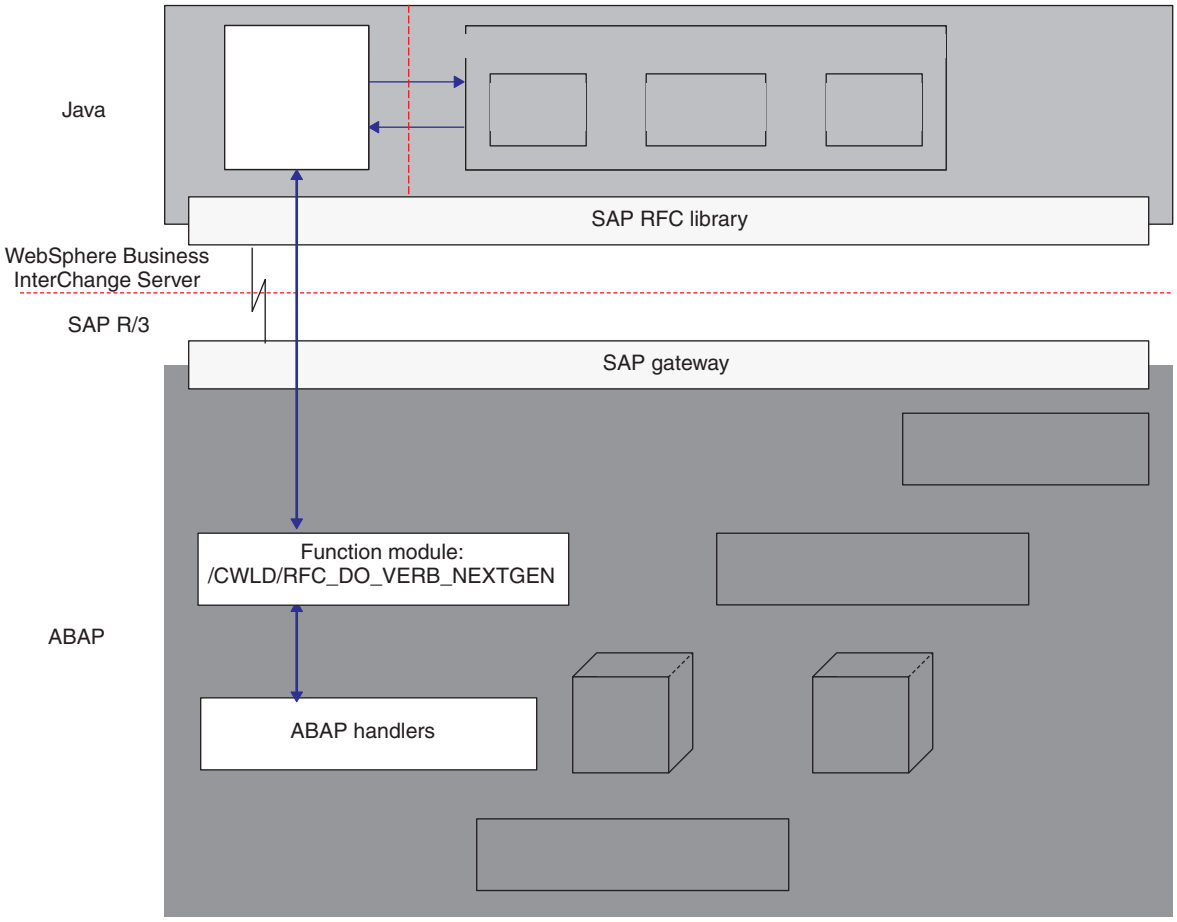


Figure 5. Business object processing of doVerbFor()

doVerbFor()

In the Java component of the connector module, the doVerbFor() method of a single business object handler implementation handles all of the business object

requests from the integration broker and all business object events from the pollForEvents() method. In either case, doVerbFor() executes in the following manner:

1. Converts an instance of a WebSphere business object for SAP to a single, predefined flat structure that contains the business object data.
2. Calls the ABAP function module /CWLD/RFC_DO_VERB_NEXTGEN, passes the business object data to it, and then waits for business object data to be returned.
3. Converts the returned business object data back into a WebSphere business object.

The doVerbFor() method passes business object data to function module /CWLD/RFC_DO_VERB_NEXTGEN and then creates an entirely new business object structure from the returned business object data.

/CWLD/RFC_DO_VERB_NEXTGEN

In the ABAP component of the connector module, the connector's ABAP function module /CWLD/RFC_DO_VERB_NEXTGEN is responsible for handling all WebSphere business object processing in the SAP application. Specifically, it routes business object data to the appropriate ABAP handler. In this sense, function module /CWLD/RFC_DO_VERB_NEXTGEN can be thought of as a business object router. It executes in the following manner:

1. Receives a business object.
2. Dynamically calls an ABAP handler to process the business object data and passes the business object data as a parameter.
3. Receives business object data from an ABAP handler and returns it to the requesting call.

/CWLD/RFC_DO_VERB_NEXTGEN uses ABAP handlers to fulfill each object type and verb-specific request. /CWLD/RFC_DO_VERB_NEXTGEN uses the value in a business object's verb application-specific information to determine which ABAP handler to call. It also checks for the archive status. /CWLD/RFC_DO_VERB_NEXTGEN can be thought of as a router from the doVerbFor() method to an ABAP handler.

ABAP handlers

ABAP handlers are unique to the connector module in that they extend the business object handler functionality from the Java component of the connector module. ABAP handlers reside in the SAP application as ABAP function modules and communicate directly with /CWLD/RFC_DO_VERB_NEXTGEN. ABAP handlers are needed to get business object data into or out of the SAP application database.

Figure 6 on page 38 illustrates the business object processing components of the ABAP Extension Module and their relationship to one another. Notice that for a single business object handler (doVerbFor()) and business object router (/CWLD/RFC_DO_VERB_NEXTGEN), there are multiple ABAP handlers.

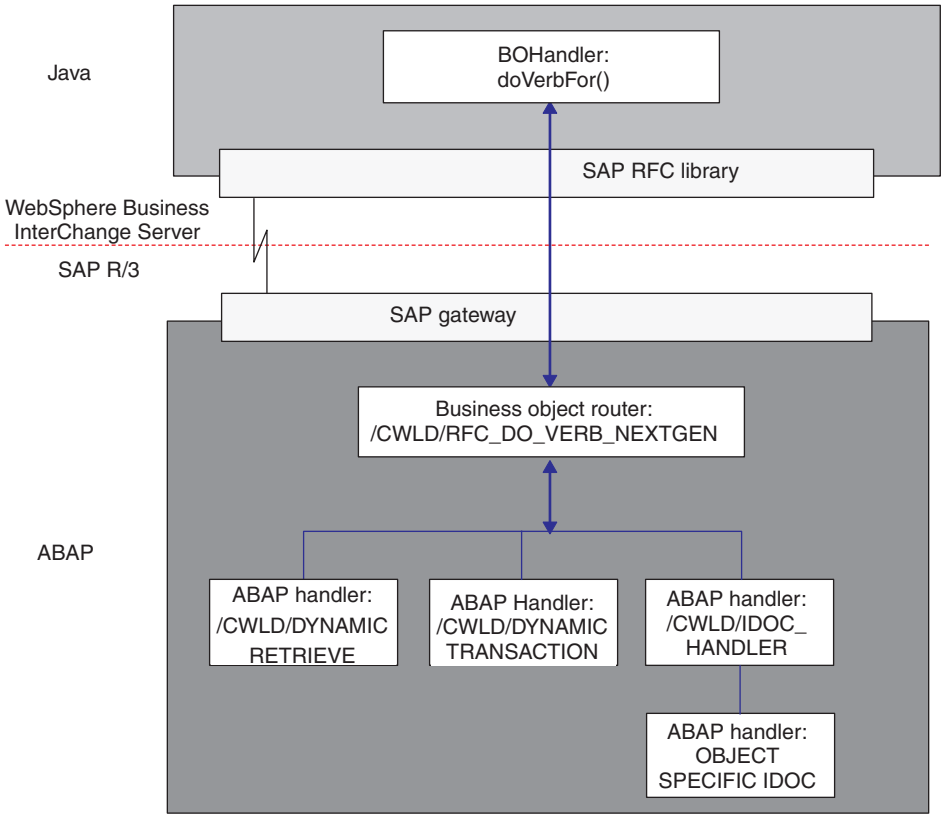


Figure 6. Adapter-provided business object processing components

ABAP handlers are responsible for adding business object data into the SAP application database (Create, Update, Delete) or for using the business object data as the keys to retrieving data from the SAP application database (Retrieve).

The adapter provides generic ABAP handlers. For example, function module `/CWLD/DYNAMIC_TRANSACTION` supports flat business objects for Create, Update, Delete, and Retrieve operations.

The WebSphere business integration system provides a metadata repository and the adapter provides a generic ABAP handler to support flat business objects. The adapter also provides an ABAP handler (`/CWLD/IDOC_HANDLER`) to support hierarchical business objects; however, you must develop an additional business-object-specific ABAP handler for each hierarchical business object that you need to support.

The WebSphere business integration system provides tools that facilitate the development process. For more information on developing business objects and ABAP handlers, see Chapter 6, “Developing business objects for the ABAP Extension module,” on page 69 and Appendix E, “Generating business object definitions using SAPODA,” on page 291.

Event notification

Event notification refers to the collection of processes that notify the connector of SAP application object events. Notification includes, but is not limited to the type of the event (object and verb) and the data key required for the external system to retrieve the associated data.

Figure 7 illustrates the event notification process, which uses the `pollForEvents()` method.

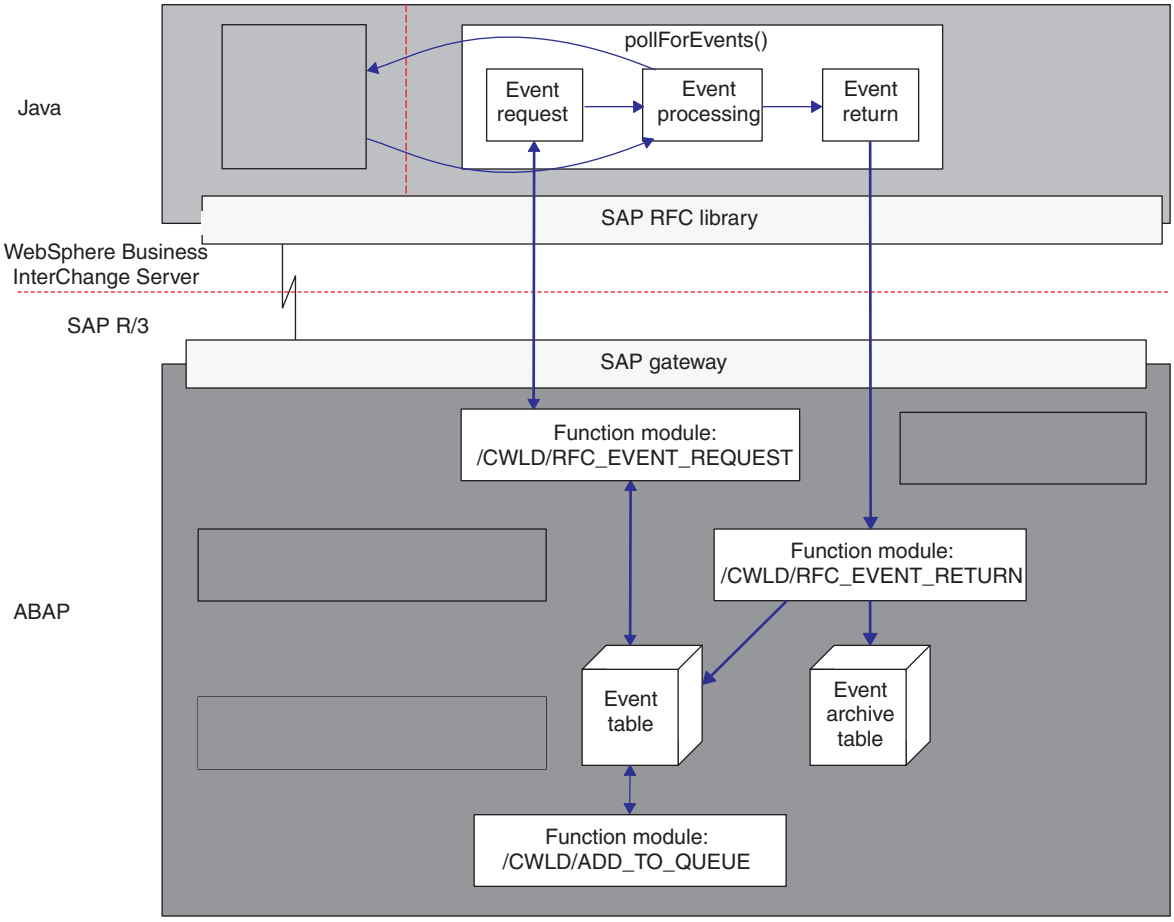


Figure 7. Event notification process

Event notification for the connector consists of two functions:

- "Event polling"
- "Event triggering" on page 42

Event polling

Event polling consists of three functions that are carried out by the `pollForEvents()` method:

- "Event request"
- "Event processing" on page 41
- "Event return" on page 41

Note: The roles of these functions are distributed in the Java and ABAP components. However, the Java component always initiates event polling.

Event request: Event request is the process of polling and retrieving events from the event table in the SAP application. The event request mechanism of the Java component has a counterpart function module in the SAP application, `/CWLD/RFC_EVENT_REQUEST`. This function retrieves events from the connector's ABAP event table, `/CWLD/EVT_CUR`.

Every triggered event enters the event table with an initial status of prequeued (status marked as P in the event table) and a default event priority of zero. Before an event can be processed, its status must be changed to queued (Q in the event table). The priority of an event must be zero before the connector retrieves the full object that it represents. For more information on event priority, see on page 44.

The status of an event changes from prequeued to queued if there are no database locks for the combination of the user who created the event and the event's key. After the event has been retrieved from the event table the status of the event is updated to event retrieved (R in the event table. If locks exist, the status of the event is set to locked (L in the event table) and the event is requeued. An ABAP constant, C_MAXIMUM_REQUEUE, defines of the number of times that an event can be requeued. If the maximum number (defaulted to 100) is attained, then the event is archived to the event archive table.

Note: Every event with a prequeued or locked status is updated with every poll. You can run into performance issues when events are triggered in batches. You can configure the polling frequency using the PollFrequency configuration property. For more information, see Appendix B, "Standard configuration properties for connectors," on page 241.

After preprocessing all prequeued events, the ABAP function module /CWLD/RFC_EVENT_REQUEST selects the events to return to the event request method in the Java component of the connector module (only events with a status of queued can be selected). The connector-specific configuration property PollQuantity (defaulted to 20) determines the maximum number of events returned for a single poll. For more information, see Chapter 2, "Installing and configuring the connector," on page 11.

The event request mechanism performs the event selection process in two steps:

1. Selects events dedicated to the connector and the integration broker.

   Events are dedicated to a specific integration broker in the event distribution table (/CWLD/EVT_DIS). The name of the integration broker specified in this table must match the name specified in the shortcut that starts the connector. For example, the standard shortcut for an SAP connector running on Windows has the format:

   ...\start_SAP.bat *SAPconnectorName integrationBrokerName* -c*ConfigFileName*

   When WMQI is the integration broker, the WebSphere business integration system identifies the integration broker specified in the event distribution table by getting values from the connector's startup command:

   - The value of the integrationBrokerName parameter links the broker instance in the startup command to the broker specified in the event distribution table.

     Note: The product's installation program uses the integration broker name specified at installation as the value of the integrationBrokerName parameter in the startup command.

   - The value of the ConfigFileName parameter identifies the Queue Manager and queues configured for the specific WMQI instance.

2. If fewer than the maximum number of events have been selected, pulls the balance from the events that are not configured for event distribution.

   For example, if the connector-specific configuration property PollQuantity is kept at 20 and there are 8 events dedicated to the specific connector and the integration broker, the mechanism selects 12 additional events.

When WMQI is the integration broker and only one Queue Manager has been configured, the names of the queues must be unique for each instance of the integration broker. When WMQI is the integration broker and a cluster has been configured, the names of the queues must be unique for each integration broker within the cluster.

If desired, you can incorporate the name of the broker (as specified in the `integrationBrokerName` parameter of the startup command) or the name of the connector into the names of the queues. For example, if two brokers are named `WMQI1` and `WMQI2`, their respective ADMINOUTQUEUEs might be named `ADMINOUTQUEUE_MQI1` and `ADMINOUTQUEUE_MQI2`, respectively.

Important: If you set up multiple connectors to poll, you must configure every event to be processed by only one connector. Otherwise the connector may send duplicate events, or may archive events instead of retrieving them.

Event processing: The event request function produces an array of events to be processed from the `/CWLD/EVT_CUR` event table. It passes these events to the event processing function, which handles them one at a time in the following manner:

1. Evaluates if the event is in the connector subscription list using the `object.verb` value.

   If an event is not in the subscription list, sets the status of the event to `not subscribed`.
2. Creates a `parentObjectOnly.Retrieve` business object if an event is in the subscription list. The event processing function sets the key value in one of the following ways:
   - If the event key value does not contain the `|Cx|` delimiter, the connector sets the value of the first key attribute to the value specified in the event key. In this case, composite keys are treated as singletons and must be interpreted by the ABAP business object processing function modules.
   - If the event key value contains one or more instances of the `|Cx|` delimiter, the connector sets the value of each specified attribute to its specified value.

   For more information about specifying a composite key for an event, see "Coding composite keys as name-value pairs" on page 95.
3. Invokes `doVerbFor()` and passes the business object data to it. Once the business object is passed, event processing waits for business object data to return.
4. Updates the status of the event array based on the `doVerbFor()` processing.
5. Delivers the business object data to the integration broker if the business object data is successfully retrieved.

Event return: After each event is processed by event request, it is returned to the SAP application using function module `/CWLD/RFC_EVENT_RETURN`. This function module makes a copy of the processed event, adds it to the event archive table (`/CWLD/EVT_ARC`), and then deletes the original entry from the event table.

Note: Events with their new status are all updated after each event is processed.

Archived events include successfully processed events, events that were processed but terminated in an error, and unsubscribed events. Each event has a status that can indicate one of the following conditions:

- The business object was successfully sent to the integration broker.
- The event produced an unknown Java return code from the connector.

- The event failed in attempting to retrieve data from the SAP application.
- The event timed out because the business object was locked.
- No collaboration subscribes to the event—relevant only when the WebSphere InterChange Server is the integration broker.

Use the IBM CrossWorlds Station tool in the SAP application to administer the event archive table. IBM CrossWorlds Station enables an administrator to display and truncate the archive table and to resubmit events for processing. For more information about maintaining the archive table and setting up log truncation, see Chapter 9, "Managing the ABAP Extension module," on page 107.

Event triggering

The connector is event-driven. In order to get events out of the SAP application, you need to implement an event triggering mechanism for each IBM WebSphere-supported business object. Event triggering for the connector comprises three functions:

- "Event detection"
- "Event triggering"
- "Event persistence" on page 45

Event detection: Event detection is the process of identifying that an event was generated in the SAP application. Typically, connectors use database triggers to detect an event. However, because the SAP application is tightly integrated with the SAP database, SAP allows very limited access for direct modifications to its database. Therefore, the event detection mechanisms are implemented in the application transaction layer above the database.

The IBM WebSphere Business Integration Adapter for mySAP.com commonly uses four mechanisms to detect an event in the SAP application:

- Code enhancements
- Batch programs
- Business Workflow
- Change Pointer

All of these event detection mechanisms support real-time triggering and retrieval of objects. In addition, code enhancements and batch programs provide functionality to delay the retrieval of events. An event whose retrieval is delayed is called a future event. For more information on triggering future events, see **"Event triggering."**

Note: Each event detection mechanism has advantages and disadvantages that need to be considered when designing and developing a business object trigger. For more information on implementing an event detection mechanism, see Chapter 7, "Developing event detection for the ABAP Extension module," on page 89.

Keep in mind that these are only a few examples of event detection mechanisms. There are many different ways to detect events.

Event triggering: Once an event is identified by one of the event detection mechanisms, it is triggered using one of the adapter-delivered event triggers.

- `/CWLD/ADD_TO_QUEUE`—function module that triggers events to the current event table for immediate processing

- /CWLD/ADD_TO_QUEUE_IN_FUTURE— function module that triggers events to the future event table to be processed at a later time

Note: Both functions are for real-time triggering. /CWLD/ADD_TO_QUEUE processes events immediately and /CWLD/ADD_TO_QUEUE processes events at a later time

If the event will be triggered in real-time, then /CWLD/ADD_TO_QUEUE commits the event to the current event table (/CWLD/EVT_CUR). Specifically, it adds a row of data for the object name, verb, and key that represents the event.

Figure 8 illustrates events triggered by /CWLD/ADD_TO_QUEUE.

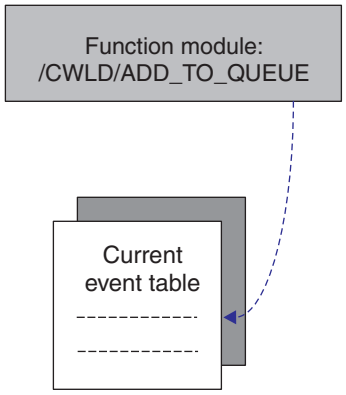


Figure 8. /CWLD/ADD_TO_QUEUE

If an event needs to be processed at a future date, then /CWLD/ADD_TO_QUEUE_IN_FUTURE commits the event to the future event table (/CWLD/EVT_FUT). Specifically, it adds a row of data for the object name, verb, and key that represents the event. In addition, it adds a Date row which is read by the adapter-delivered batch program /CWLD/SUBMIT_FUTURE_EVENTS. This batch program can be scheduled to retrieve events from the future event table. Once it retrieves an event, it calls /CWLD/ADD_TO_QUEUE to trigger the event to the current event table.

Note: /CWLD/ADD_TO_QUEUE_IN_FUTURE uses the system date as the current date when it populates the Date row of the future event table.

Figure 9 illustrates events triggered by /CWLD/ADD_TO_QUEUE_IN_FUTURE.

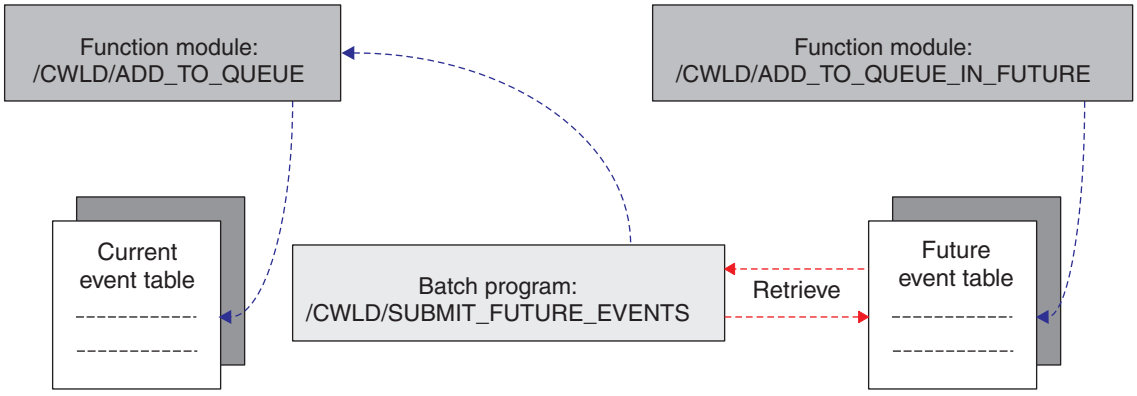


Figure 9. /CWLD/ADD_TO_QUEUE_IN_FUTURE

For more information on triggering events for the future event table, see Chapter 7, "Developing event detection for the ABAP Extension module," on page 89.

All events are added to the current event table using /CWLD/ADD_TO_QUEUE. In addition to adding a row of data to the current event table, /CWLD/ADD_TO_QUEUE can be set up for:

- Event filtering
- Event distribution
- Event priority

Event filtering, event distribution, and event priority are executed as part of the event trigger and by no other program. They result in either the event's restriction (filtering), or modification (event distribution and event prioritization).

Event filtering The event trigger can be used to filter out events that you do not want added to the event table. The adapter provides an ABAP include program (/CWLD/TRIGGERING_RESTRICTIONS) that enables you to restrict specific events for this purpose.

Event distribution Load balancing can be used to distribute event processing across multiple connectors allowing you to process multiple events at the same time. The event trigger provides this capability through the event distribution table (/CWLD/EVT_DIS). You can dedicate business objects to be retrieved by a specific connector. Also, event distribution can take a single event and replicate it one or more times for each subscribed combination of connector and integration broker.

Attention: If you are using multiple connectors to poll, you must dedicate every subscribed event to a specific connector. Failure to do so may result in duplicate events delivered. You must guarantee that there is no dependency between objects dedicated to different connectors, because this may result in events being delivered out of sequence.

For example, assume you have a single integration broker named CrossWorlds1 that subscribes to two different business objects, BO_A and BO_B. The BO_A business object is small and can be retrieved quickly whereas BO_B is large and takes much longer to retrieve. With two connectors polling, SAP1connector and SAP2connector, you can set up the event distribution table so that SAP1connector retrieves BO_A and SAP2connector retrieves BO_B. SAP1connector can continuously poll small objects of type A, while SAP2connector focuses on the larger type B objects.

Note: For information on how the WebSphere business integration system identifies each unique instance of a WMQI integration broker, see "Event request" on page 39.

Important: If the event distribution table is not configured for a specific object, then each event triggered for that object is available for any combination of connector and integration broker.

Event priority You can set the event priority for each combination of business object, connector, and integration broker by delaying the retrieval of events. An

event's priority indicates the number of polls that are needed before the event is picked up for delivery. For example, if you set the priority of an event to 10, then the connector polls the event table ten times before the event is retrieved. Each time the connector polls, the priority value is reduced by one until it reaches zero.

By default, all events are given a priority of zero. An object's priority is configured in the same ABAP table as event distribution.

Figure 10 illustrates the event triggering functionality inside the SAP application. The events E1, E2, and E3 are received by the event trigger `/CWLD/ADD_TO_QUEUE`. E1 represents a Customer event and E3 represents an Order event. Event distribution is set up so that all Customer objects are handled by SAP1connector and all Order objects are handled by SAP2connector. In this environment, both connectors use the same integration broker. Because E1 is a Customer object, it is polled by SAP1connector and because E3 is an Order object, it is polled by SAP2connector. E2 is an Inventory object that is filtered out by code in the restriction program `/CWLD/TRIGGERING_RESTRICTIONS` that restricts inventory objects to a specific warehouse.

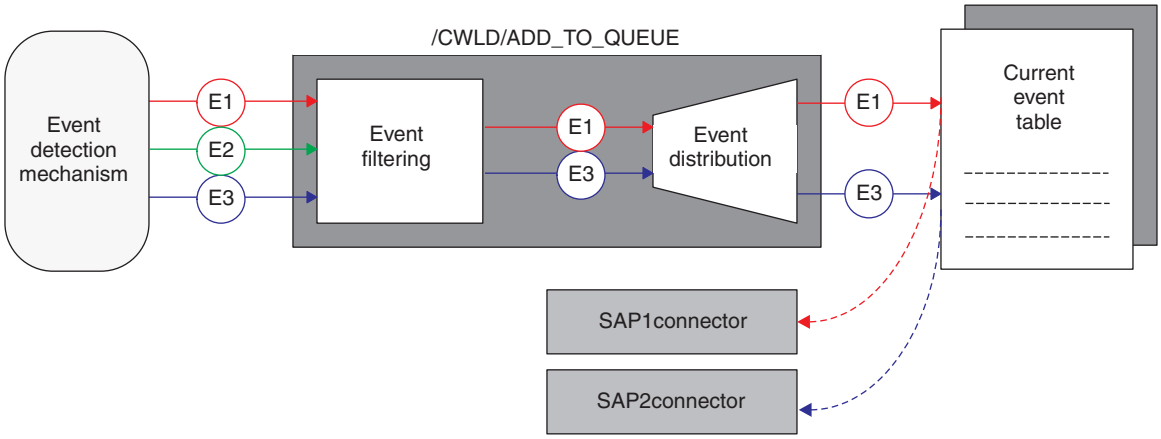


Figure 10. Event priority with function module /CWLD/ADD_TO_QUEUE

Event persistence: Once the event trigger inserts an event into the event table, the event is committed to the database with its event distribution and event priority values set. At this time, only polling can modify the event. When the event polling processes is completed, meaning the event was retrieved from the SAP application and processed by the Java component of the connector, a copy of the processed event is added to the event archive table (`/CWLD/EVT_ARC`). The original event is then deleted from the event table.

Note: You can resubmit an event from the archive table. Keep in mind that the event is simply moved to the event table and is not triggered again. Specifically, it does not pass back through event filtering, event distribution, and event priority.

Chapter 4. Installing and customizing the ABAP Extension module

This chapter describes the installation and customization of the ABAP Extension Module only and assumes that you have already installed and configured the connector. For more information on installing and configuring the connector, see Chapter 2, "Installing and configuring the connector," on page 11 and Appendix B, "Standard configuration properties for connectors," on page 241. Customizing the connector is optional, but recommended.

This chapter contains the following sections:

- "Connector transport file installation"
- "Verifying connector transport file installation" on page 51
- "Enabling the SAP application for the connector" on page 53
- "Modifying adapter-delivered ABAP objects" on page 55
- "Preventing event ping-pong" on page 55

All of the components of the connector can be found in the `\connectors\SAP` directory for Windows and the `/connector/SAP`and `/bin` directories for UNIX. The transports are installed on an SAP R/3 application or database server as described below in "Connector transport file installation."

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.

Connector transport file installation

The transport files for the IBM WebSphere Business Integration Adapter for mySAP.com (SAP R/3 Version 4.x) contain a variety of objects, such as table structures, functions, and data. These development objects must be imported into your SAP installation to provide specific functionality required by the ABAP Extension Module.

Each transport file is included in a `.zip` file. For example, the transport file for the SAP R/3 version 4.x Primary transport is located in the `4_Primary.zip` file.

Once the required transport files have been successfully loaded, the business object-specific transports can be loaded in any order. See the transport note included in each transport `.zip` file for detailed information about the transport file.

Creating the namespace for connector transport installation

Create the namespace for the connector before installing the connector transport files. This step is mandatory for SAP R/3 version 4.0, as some of the transports will fail if the namespace is not created.

Note: You must create the connector's namespace prior to modifying one of the connector's ABAP objects in any SAP R/3 application version 4.x.

Creating the /CWLD/ namespace

1. Open the Workbench Organizer: Tools window (transaction SE03).
2. Expand the Administration menu and double-click on the Display/change namespaces option.
3. Click the Display->Change button (Ctrl+F1).
4. Click the Continue button to close the Information window.
5. Click the New entries button (F5) and type `/CWLD/` in the Namespace field.
6. Select the Namespace role field, expand it (F4) to see options, and then select Recipient (C).
7. Type `CrossWorlds Namespace` in the Short text field and type `CrossWorlds` in the Owner field. Click the save button (Ctrl+S). If your system is set up to track customizing changes, you will be prompted for a change request which will allow you to transport the namespace to another system.

Making the namespace available for modifications

ABAP objects in the connector's namespace cannot be modified until you make the Namespace available for modification. To update SAP4.x delivered ABAP objects, you must have a repair license to modify the objects. Contact IBM technical support to obtain the license.

1. Open the Workbench Organizer: Tools window (transaction SE03).
2. Expand the Administration menu and double-click on the Display/change namespaces option.
3. Click the Display->Change button (Ctrl+F1).
4. Click the Continue button to close the Information window.
5. Double-click on `/CWLD/` and enter the repair license. Click the Save button (Ctrl+S).
6. Click the Back button (F3) twice, expand the Administration menu and double-click on the `Set system change` option.
7. Place a check mark in the Modifiable column of the Namespace row. Click the Save button (Ctrl+S).

Connector transport files

The IBM WebSphere Business Integration Adapter for mySAP.com includes seven connector transport files. Modifications required by the adapter are handled by these connector transport files. The Primary, Utilities, and Request transport files are required.

The following is a list of the SAP R/3 version 4.x connector transport files. To ensure that all necessary tables are created before the data for those tables is added, you must install the transports in the order listed. These files can be found in `\connectors\SAP\dependencies\transports_4x`.

- `4_Primary`
- `4_Utilities`
- `4_Request`
- `4_Delivery`
- `4_NumberRange`
- `4_Tools_Maintenance`
- `4_Tools_Development`

The functionality provided by the Primary, Utilities, Request, and Delivery transport files make up the runtime components. The Tools_Maintenance and Tools_Development transport files can be installed at anytime after the required transport files are installed. They are not required for your runtime environment.

4_Primary

This transport file contains the development objects, which should be loaded only once into the system. It contains the number range objects, the development classes, the dynamic transaction declaration include program as well as the restriction include program, which can be used to make customer-specific changes to the triggering logic.

Important: If you apply this transport file to a system that already has the connector running on it, the contents of the transport file may overwrite changes that were made to the existing environment.

4_Utilities

This client-independent transport file contains objects and functionality that are shared among the request, delivery, development, and maintenance components. For example, it contains the log and data elements.

4_Request

This client-independent transport file contains the functionality required to support business object request operations.

4_Delivery

This client-independent transport file contains the functionality required to support event delivery operations including event triggering and event polling.

4_Number Range

This client-dependent transport file contains the four number ranges in their initial state. You can use these intervals or create the number range intervals themselves.

Attention: Reimporting the Number Range transport file initializes your existing number range intervals for the connector. This corrupts the data in the connector's log, current event, future event, and archive tables if those tables are not refreshed before reusing.

This transport file can be installed at any time after the Primary transport has been installed.

4_Tools_Maintenance

This client-independent transport file contains the functionality required to support maintenance operations such as displaying the log statistics and event tables.

4_Tools_Development

This client-independent transport file contains the functionality required to support the development of objects.

Installing connector transport files

The connector transport files make all necessary modifications to SAP by importing programs and other development objects delivered with theIBM WebSphere Business Integration Adapter for mySAP.com. They do not alter any SAP programs or modify user exits.

Attention: If you are reapplying transports, note that this resets your environment. Any development done prior to reapplying the transport files will be overwritten.

In the following instructions, `SID` refers to the SAP system ID, and <*TransportFileName*> refers to the name of the transport file. However, the characters that make up the transport file name appear in a different order in the installation directory from the way the name is passed as a parameter to the various `tp` commands. In the `\usr\sap\trans\cofiles` directory, the format of a transport file name is `K9`*xxxxx*`.`*SID*, but when the filename is passed as a parameter it has the format *SID*`K9`*xxxxx*. For example, the file name `K912345.D30` is passed as a parameter as `D30K912345` because D30 is the SID of the source system.

Attention: Do not change the names of the connector transport files.

To install the transports:

1. Log in as the SAP administrator, <`SID`>`adm`.
2. Copy the transports to the SAP database server. There are two kinds of transport files:
   a. Copy files that have names beginning in `K` to the `\usr\sap\trans\cofiles` directory.
   b. Copy the other files to the `\usr\sap\trans` data directory.
3. Check the connection to the database and determine the path of the `tpparam` file by running the `tp connect` command:

   ```
   tp connect <SID>
   ```

   If this command fails, try adding the path of the `tpparam` file as a second parameter:

   ```
   tp connect <SID> pf = <path of tpparam>
   ```

   For example, if the SID is `P11` and the path of the `tpparam` file is `\usr\sap\trans\bin\tpparam`, the command is:

   ```
   tp connect P11 pf = \usr\sap\trans\bin\tpparam
   ```

   If `tp connect` succeeds when you specify the path of the `tpparam` file and fails when you do not, you should specify the optional `tpparam` path in the commands described in step 3.
4. Import the transport files in one of the following two ways:
   - "Use adapter-delivered commands"
   - "Use an SAP transaction code" on page 51

Use adapter-delivered commands

In `\usr\sap\trans\bin`, execute the following commands for each transport, in the order specified:

```
tp addtobuffer <TransportFileName> <SID> pf = tpparamFilePath
tp import <TransportFileName> <SID> u023689 CLIENT=<CLIENT#> pf = tpparamFilePath
```

Use an SAP transaction code

In the Transport Management s ystem (transaction STMS):

1. Click the Import overview icon (F5).
2. Double-click the appropriate queue to be updated.
3. In the menu bar, click Extras, then click Other requests, and then click Add.
4. Populate the transport request field, and then click the check mark (enter).
5. When the Add Transport Request confirmation window appears, click Yes to attach the import to the queue.
6. Place the cursor on the transport that was just added.
7. In the menu bar, click Request, and then click Import.
8. Populate the Target client field, and click the check mark to import the transport file.

You must install the transports in the order listed in "Connector transport file installation" on page 47.

After the transports are installed, change the development class to follow the migration path of your development classes. Use IBM CrossWorlds Station (transaction `/n/CWLD/HOME`) to do the following:

1. Click the Tools tab, and then click the Transport Layer button.
2. Select the appropriate Transport layer entry, and then click the Save button.

Attention: Any changes you make to development objects that were in the connector transports should be well documented outside of SAP. Changes can be overwritten by the next release of transport files. If changes are overwritten, they must be reapplied manually. For more information on upgrade issues, see Chapter 10, "Upgrading the ABAP Extension module," on page 115.

Verifying connector transport file installation

To verify that the connector transport files are installed properly, you must:

- "Verify that transport files were moved to the SAP application"
- "Verify that SAP generated the objects successfully" on page 52

Verify that transport files were moved to the SAP application

To verify that the connector transport files were physically moved into the SAP application, examine the transport logs in one of the following ways:

- Use the Transport Organizer (transaction SE01)
- Use the Transport Management System graphic interface (transaction STMS)

Using the Transport Organizer (transaction SE01)

To use the transport organizer (transaction SE01):

1. Populate the number field with the name of the transport file.
2. Click Display to see the log.

Using the Transport Management System graphic interface (transaction STMS)

To use the Transport Management System graphic interface (transaction STMS):

1. Click the Import overview icon (F5).
2. Double-click the appropriate queue.
3. Right-click the transport number, and then select Logs.

4. Examine the log to see if the installation was successful.

Verify that SAP generated the objects successfully

To verify that SAP generated the objects successfully:

1. Go to transaction SE38
2. Enter the program `/CWLD/CONSTANTS`.
3. Select Source Code, and then click Display.
4. From the Program menu, click Generate.
5. Click Select All, and then click Continue (F2).

This generates all of the adapter programs that include these programs.

If you get the response `Programs successfully generated`, you can assume that the transport was successful.

Upgrading the ABAP Extension Module

Upgrading the connector involves installing the latest adapter files and loading the latest ABAP transport files for the ABAP Extension Module. For additional upgrade information, see the *System Installation Guide for UNIX* or *for Windows*.

It is recommended that you backup your current connector files (for example the configuration and message files `CN_SAP.txt` and `SAPConnector.txt`) before starting. Before loading the connector definition into the repository, you may want to remove all supported object references except those you need.

Attention: If you install the latest Primary and Number Range transport files, you will overwrite your existing number range interval information. Overwriting the number range intervals may cause existing events and objects to be out of sync because the number range interval is reset to zero (0).

After you install the latest version of the connector, install the latest ABAP transport files for the version of SAP you are going to use. Without these, you cannot transport the existing components of the ABAP Extension Module. Transport installation is described in "Connector transport file installation" on page 47. Keep in mind that installing new transport files overwrites any modifications to adapter-delivered code.

Make sure that you use the correct transport files for your environment. For example, if you are using an SAP R/3 version 4.6 environment, install the adapter's 4.x transport files. Doing so ensures that any warning or errors you get when you load your objects are in relation to the SAP R/3 version 4.x environment and not anything else that was brought over from SAP R/3 version 3.x. This allows you to resolve problems that relate to differences in SAP R/3 version 4.x.

After you install the latest version of the connector and the latest ABAP transport files for the version of SAP you are going to use., configure the new connector.

If your are upgrading from the C++ version of the connector to the latest Java-based version, you need to configure the new connector-specific configuration properties "Modules" on page 23 and "Namespace" on page 23. For more information on these properties, see "Connector-specific configuration properties" on page 18.

Enabling the SAP application for the connector

After installing the connector and configuring the standard and connector-specific configuration properties, you have the option of modifying the event handling and logging capabilities for the connector from within the SAP application.

Setting up event distribution

Load balancing distributes event and business object request processing across multiple connectors. The IBM WebSphere Business Integration Adapter for mySAP.com can handle only one transaction at a time. However, you can process multiple events and business objects at the same time if you set up multiple connectors to handle specific business objects. For more information on setting up multiple connectors, see "Installing multiple connectors" on page 16.

To set up event distribution for multiple connectors:

1. Go to IBM CrossWorlds Station (transaction `/n/CWLD/HOME`).
2. Click the Configuration tab, and then click the Event Distribution button.
3. Click the New Entries button (F5), and in the New Entries window, enter the business object name, connector name, and integration broker name.
4. Enter a number in the counter field for each business object. The combination of the business object and counter provides a unique key for the event distribution table. The counter can be any number up to six digits.

Note: In a test environment, you may have multiple users testing the same business object that is subscribed to by multiple connectors. If each user wants only a certain event for that business object, then you can specify a user name to differentiate between which event is passed to the combination of which connector and which integration broker. In the User (Event Trigger) field, enter the appropriate user name for the business objects. For information on how the WebSphere business integration system identifies each unique instance of a WMQI integration broker, see "Event request" on page 39.

Setting up event filtering

The configuration table in the SAP application cannot accommodate all modifications. Therefore, the adapter provides an ABAP include program that can be modified to filter events. This program, `/CWLD/TRIGGERING_RESTRICTIONS`, is called from within the event trigger `/CWLD/ADD_TO_QUEUE`, to enable additional filtering of events.

Note: You must have developer privileges to make changes because the code needs to be recompiled.

To view or modify the include program `/CWLD/TRIGGERING_RESTRICTIONS`:

1. Go to IBM CrossWorlds Station (transaction `/n/CWLD/HOME`).
2. Click the Configuration tab, and then click the Event Restriction button.

Setting up event priority

You can set the priority of an event to be processed based on its importance. By setting the priority of each combination of business object, integration broker, and connector, you can delay a connector's retrieval of an event. For example, if you set the priority of an event to 10, the connector polls the event table ten times before retrieving the event. So, if the connector polls the event table every 5 seconds, the

connector picks up the event after 50 seconds. Each time the connector polls, the priority value is reduced by one until the event is retrieved and processed.

To set the priority of an event:

1. Go to IBM CrossWorlds Station (transaction /n/CWLD/HOME).
2. Click the Configuration tab, and then click the Event Distribution button.
3. Fill in the Priority column with a value between 1 and 99 for the appropriate business object.

Increasing log tablespace size

The connector's log tables are located, by default, in the tablespace named PSAPUSER1D, and the indexes are located in tablespace PSAPUSER1I. The PSAPUSER1D and PSAPUSER1I SAP application tablespaces, which are reserved for customer use, are typically small. Because of the default size, these tablespaces can fill up quickly, depending on the level of activity and the logging level of the SAP installation for the adapter.

To view the current size of these tablespaces, go to transaction DB02, and then click the Current Sizes button. The volume of events captured by the WebSphere business integration system determines the size needed for these tablespaces.

If the default sizes are too small, ask the SAP database administrator to modify them.

Verifying number ranges for transport objects

There are four objects for the WebSphere business integration system that must have an adequate number range within the SAP application. When the transports are installed, the following objects and their default number ranges are set:

- /CWLD/EVT
- /CWLD/IDOC
- /CWLD/LOG
- /CWLD/OBJA

Verify that the associated number ranges are set correctly. To view the number ranges:

1. Go to transaction SNRO.
2. Populate the Object field with the object name (for example, /CWLD/EVT).
3. Click Number Ranges, and then click Intervals.

Attention: If you reinstall the 4_Primary or 4_Delivery connector transport in an installation where events have already been generated, new events may be created using existing event IDs. To prevent this problem, turn off logging by going to the Configuration tab in IBM CrossWorlds Station, and then truncate the log completely before reimporting the connector transport file. Once the connector transport file has been successfully loaded, turn logging back on. For more information on truncating the event log, see "Setting up truncation of the event log" on page 110.

Modifying adapter-delivered ABAP objects

All adapter objects such as tables, function modules, and programs use the product namespace /CWLD/. If you need to modify adapter-delivered ABAP code, you must obtain a modification key by contacting IBM technical support.

Preventing event ping-pong

Ping-pong occurs when the successful execution of a request to an application triggers an event in that application that results in an event being created in the event table. If the connector is set to poll the event table, it picks up the new event, sends it back to the original source application, which in turn triggers its own additional event. This new event in the source application continues the cycle.

Note: When the WebSphere InterChange Server is the integration broker, collaboration look-up and cross-reference mapping should prevent a duplicate record from being created in the source application. However, this additional processing is unnecessary.

To prevent ping-pong with the connector, do the following:

1. Go to IBM CrossWorlds Station (transaction /n/CWLD/HOME).
2. Click the Tools tab, and then click the Config Objects button.
3. Click the New Entries button and enter the following information:

   Configuration name—`Trigger: NoEventForUser`

   Text—`Prevent triggering for certain users`
4. Return to the Configuration Tab in IBM CrossWorlds Station, and then click the Configuration Values button.
5. Click the New Entries button, and then add an entry for each *User Id* for which you want to prevent triggering of events.
   - Configuration Name—`Trigger: NoEventForUser`
   - Counter—*Any Number*
   - Configuration value—*User Id (connector name)*

Note: Keep in mind that this prevents the connector from triggering events.

Chapter 5. Business object processing in the ABAP Extension module

This chapter discusses business object processing for the ABAP Extension Module. It provides a detailed description of how the connector processes business objects. The chapter is set up to show the progression of a business object through the Java and ABAP components of the connector.

This chapter contains the following sections:

Business object processing for the extension module of the IBM WebSphere Business Integration Adapter for mySAP.com (SAP R/3 Version 4.x) is the same for all business objects regardless of the specific native SAP API that is used. For example, if you develop a business object based on a Call Transaction or an IDoc, the business object data is processed the same way. The processing is the same whether a business object is sent into the SAP application as a retrieve performed as part of event notification or as a business object request. The business object's verb also does not change the processing.

Figure 11 illustrates the conversion and processing of an application-specific business object to a flat structure and then back to an application-specific business object. Note that the business object data that is passed out of the SAP application must have the same structure as the data passed in, but the data might not have the same values.

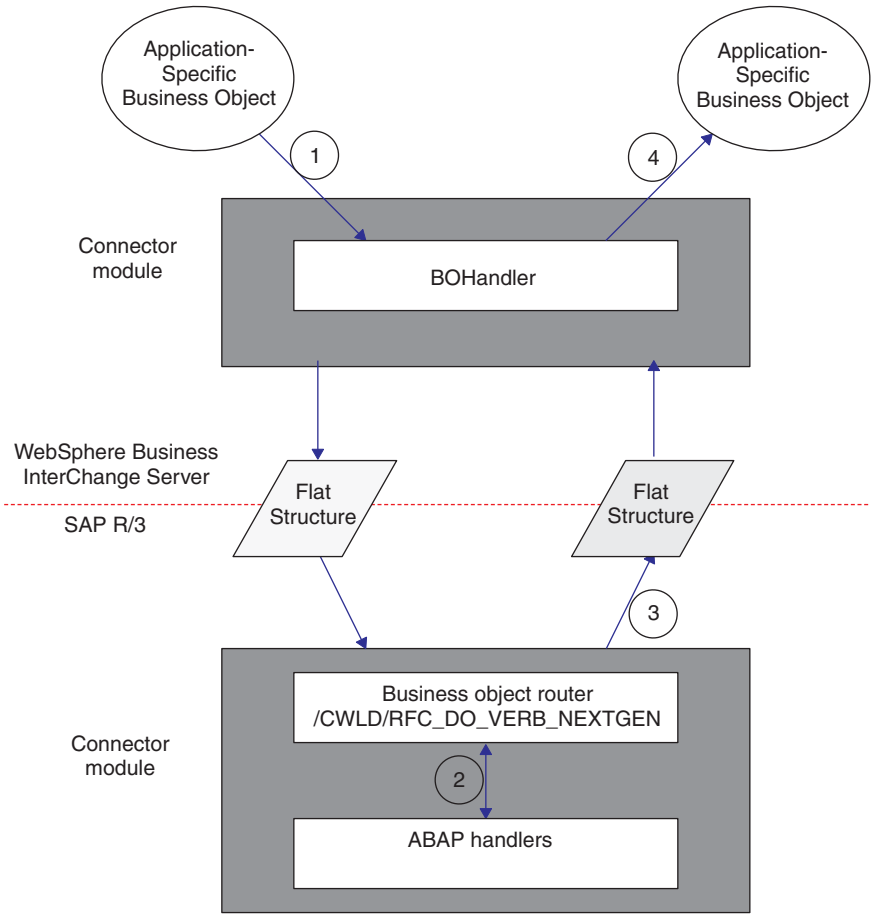


Figure 11. Business object processing

Business object processing consists of four steps. The four steps listed below correspond to the numbers in Figure 11.

1. The connector converts an application-specific business object into a flat structure containing business object data and passes the data to the SAP application.
2. The connector's function module `/CWLD/RFC_DO_VERB_NEXTGEN` dynamically routes the business object data to an ABAP handler.
3. The ABAP handler processes the business object data, generates business object response data, and returns new business object data to the connector through `/CWLD/RFC_DO_VERB_NEXTGEN`.
4. The connector receives the new business object data, and uses it and the business object definition of the application-specific business object to create a new business object to pass to the integration broker.

Business object conversion to a flat structure

As a first step in business object processing, the connector converts a business object into a flat structure that can be processed in the SAP application. The format of the flat structure is the same for all types of business objects (such as Call Transaction-based or IDoc-based business objects). The flat structure is reformatted data from an application-specific business object. The only difference between the

two forms of data is that the flat structure does not maintain parent and child business object relationships. Therefore, the connector relies on a set of rules to create a flat structure.

When converting a business object into a flat structure, the connector creates a structure in memory and then populates it with data from the business object. In doing so, it passes the following data into the SAP application from the business object:

- Business object name
- Business object application-specific information
- Business object verb
- Business object verb application-specific information
- Attribute name
- Attribute property `IsKey`
- Attribute property `AppText`
- Attribute value

Table 8 shows the generic flat structure of a business object. The connector uses this flat structure when adding the business object data from a WebSphere business object.

Table 8. Generic flat structure representation of a WebSphere business object for SAP

Field name	Data type	Length	Description
ATTR_NAME	CHAR	32	Attribute Name (example, CustomerId)
BLANK1	CHAR	1	Delimiter
ATTR_VALUE	CHAR	200	Attribute Value (example, 00000103)
BLANK2	CHAR	1	Delimiter
ISKEY	CHAR	1	1= true, 0 = false; attributes only
BLANK3	CHAR	1	Delimiter
ISNEW	CHAR	1	1 = BO; 0 = verb or attribute
BLANK4	CHAR	1	Delimiter
PEERS	CHAR	6	Indicates number of peers of an array of business objects
BLANK5	CHAR	1	Delimiter
OBJ_NUMBER	CHAR	6	Not used
BLANK6	CHAR	1	Delimiter
APPTEXT	CHAR	120	Application-specific information of object, verb or attribute
BLANK7	CHAR	1	Delimiter

Note: The BLANK*n* field names always contain a single character (CHAR) space and should never be populated.

In order for the data conversion to work properly, the business object data in the flat structure must strictly adhere to a set of rules. These rules are defined in this initial data conversion step:

- Each business object attribute is placed sequentially into a flat structure, where one row corresponds to one attribute.
- Hierarchical business objects are converted as depth and then breadth.

When the connector populates the flat structure with business object data, the connector loops through each business object twice, beginning with the top-level business object.

1. In the first pass, it sets all simple attributes. Each attribute equals one row in the flat structure.
2. In the second pass, it recursively executes the same processing in step 1 for each child business object.

Attributes that represent child business objects are not included in their parents. Instead, each child that contains data is created as a complete business object. The result is a single list of attributes ordered by depth, then breadth.

Figure 12 illustrates the data conversion of a WebSphere business object for SAP into a flat data structure. The conversion of data always follows the rule of depth first and then breadth. In the example, the top-level parent business object, SAP_Order, has two children, SAP_LineItem (1) and SAP_LineItem (2), which are considered peers. SAP_LineItem (1) has one child business object, SAP_ScheduleLines.

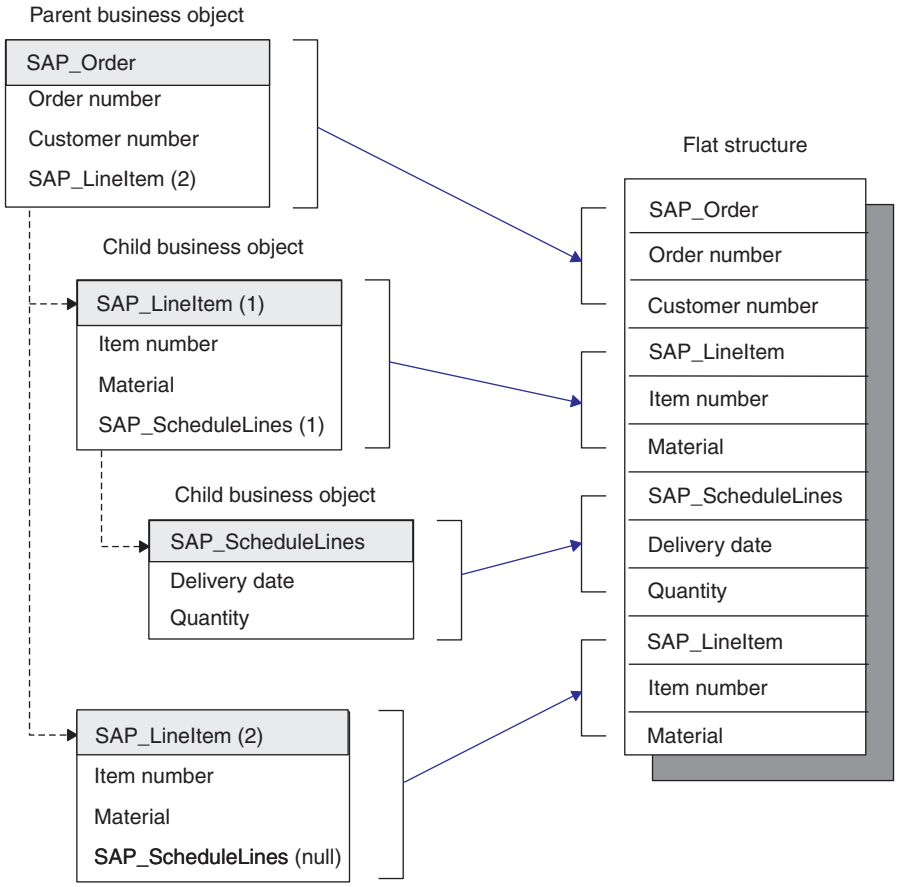


Figure 12. Conversion from a business object to a flat structure

It is important to understand the ordering of the business objects and their attributes when designing a business object definition. The following tables illustrate the result of the conversion of an WebSphere business object to a flat structure. Table 9 on page 61 represents a flat structure for a flat business object, SAP_Material, whose key value is ItemID. In this example, there is no

application-specific information for the business object or any of the attributes. Table 10 represents a flat structure of a hierarchical business object based on an IDoc Sales Order.

Table 9. Flat business object SAP_Material

ATTR_NAME	ATTR_VALUE	ISKEY	ISNEW	PEERS	OBJ_ NUMBER	APPTEXT
BoName	SAP_Material	0	1	1	(blank)	(blank)
BoVerb	Retrieve	0	0	1	(blank)	:/CWLD /DYNAMIC_RETRIEVE
ItemID	000000000000001179	1	0	1	(blank)	(blank)
ShortDesc	CxIgnore	0	0	1	(blank)	(blank)
ObjectEventID	SAP_124	0	0	1	(blank)	(blank)

In this example, there is no application-specific information for the business object or any of the attributes.

Table 10. Hierarchical business object based on an IDoc sales order

ATTR_NAME	ATTR_VALUE	ISKEY	ISNEW	PEERS	OBJ_ NUMBER	APPTEXT
BoName	SAP_Order	0	1	1	(blank)	YXRV4B01
BoVerb	Create	0	0	1	(blank)	[*archive:methods*]
Currency	USD	0	0	1	(blank)	E1EDK01:CURCY
OrderId	CxIgnore	1	0	1	(blank)	E1EDK01:BELNR
ObjectEventId	SAP_124	0	0	1	(blank)	E1EDK01: ObjectEventId
BoName	SAP_LineItem	0	1	2	(blank)	Z1XRV40
BoVerb	Create	0	0	2	(blank)	(blank)
Createdby	User1	1		2	(blank)	Z1XRV40:ERNAM
ObjectEventId	SAP_125	0	0	2	(blank)	Z1XRV40: ObjectEventId
BoName	SAP_ ScheduleLines	0	1	1	(blank)	E1EDK14
BoVerb	Create	0	0	1	(blank)	(blank)
Qualifier	001	1	0	1	(blank)	Z1XRV40:QUALF
OrganizationId	1000	0	0	1	(blank)	E1EDK14:ORGID
ObjectEventId	SAP_126	0	0	1	(blank)	E1EDK14: ObjectEventId
BoName	SAP_LineItem	0	1	2	(blank)	Z1XRV40
BoVerb	Create	0	0	2	(blank)	(blank)
Createdby	User1	1	0	2	(blank)	Z1XRV40:ERNAM
ObjectEventId	SAP_127	0	0	2	(blank)	Z1XRV40: ObjectEventId

The first two rows, `BoName` and `BoVerb`, are added by the connector for each business object. `BoName` and `BoVerb` are keywords that cannot be used as business object attributes.

Business object data routing to ABAP handlers

Once the business object data is converted into a flat structure, the business object data is passed into SAP memory by calling the adapter's ABAP function module `/CWLD/RFC_DO_VERB_NEXTGEN`. `/CWLD/RFC_DO_VERB_NEXTGEN` does not manipulate the business object data; it simply routes it to the appropriate ABAP handler for

further processing. After /CWLD/RFC_DO_VERB_NEXTGEN passes the business object data to an ABAP handler, it waits for business object data to be returned.

Note: Remember that every business object retrieve and request is processed through /CWLD/RFC_DO_VERB_NEXTGEN.

/CWLD/RFC_DO_VERB_NEXTGEN uses a business object's verb application-specific information to determine which ABAP handler processes the business object data. At runtime, /CWLD/RFC_DO_VERB_NEXTGEN reads the verb application-specific information and passes the business object data to the specified ABAP handler.

Every ABAP handler must reserve the use of verb application-specific information for the connector. The format for the verb application-specific information is:
:function1:function2:function3

where /CWLD/RFC_DO_VERB_NEXTGEN executes *function1*, passing *function2* and *function3* as parameters. For example, Customer Update and Material Retrieve execute only function1:

For Create, Update, and Delete verbs, specify :/CWLD/RFC_DYNAMIC_TRANSACTION

For the Retrieve verb, specify :/CWLD/RFC_DYNAMIC_RETRIEVE

One of the ABAP handlers provided by the adapter is function module /CWLD/IDOC_HANDLER. This ABAP handler reformats the data of the flat structure into an instance of an IDoc definition and passes that reformatted data to another ABAP handler written to handle that specific type of IDoc. The following examples illustrate the use of the IDoc Handler API:

Sales Order Update = :/CWLD/IDOC_HANDLER:Y_XR_ORDER_C2

Sales Order Retrieve = :/CWLD/IDOC_HANDLER:Y_XR_ORDER_C4

In the examples, /CWLD/IDOC_HANDLER is executed and passes the second function module name as well as the business object data. /CWLD/IDOC_HANDLER executes the call to the second ABAP handler to pass the business object data in an IDoc format to the Y_XR_ORDER_C2 or Y_XR_ORDER_C4 function module written specifically to handle Order objects. For steps on setting up verb support for the IDoc handler, see "Developing business object definitions using object definition generator" on page 74.

Note: /CWLD/RFC_DO_VERB_NEXTGEN uses the value of *function1* only. *function2* and *function3* may be used by the ABAP handler.

To dynamically call an ABAP handler, /CWLD/RFC_DO_VERB_NEXTGEN requires the interface of every ABAP handler to be exactly the same. This enables /CWLD/RFC_DO_VERB_NEXTGEN to send and receive business object data, as well as a return code and a return text message to any ABAP handler. For more information on the functional module interface, see "IBM WebSphere function module interface" on page 71.

How ABAP handlers process business object data

The function of an ABAP handler is to get business object data into or out of the SAP application database. When processing business object data, ABAP handlers:

1. Interpret business object data.
2. Integrate data with SAP native APIs.
3. Reformat all data returned from native APIs.

Business object data and ABAP handlers

Every ABAP handler receives business object data in the same format (flat structure). However, each ABAP handler has specific requirements for business objects that are determined by the complexity of the WebSphere business object definition, the native API that SAP provides, and the level of functionality that the ABAP handler provides. For these reasons, ABAP handlers may interpret business object data by parsing it into a structure specific to the business object. This enables the ABAP handler to more easily manipulate the data.

Note: Parsing the data is not required. However, it simplifies the ABAP handler's processing of a business object.

The adapter provides several ABAP handlers, such as an IDoc handler. The IDoc handler leverages SAP's IDoc technology by providing an ABAP handler to interpret business object data by reformatting it into an IDoc-based structure for the ABAP handler to use.

Business object data and SAP native APIs

Once the ABAP handler interprets the business object data, the ABAP handler must integrate it with the SAP application database. It must manipulate the business object data to use SAP native APIs such as Call Transaction or ABAP SQL to get data into or out of the application database.

Create, update, and delete processing

The intent of a Create, Update, or Delete operation is to modify the SAP application database. While the SAP application database schema for a given business object defines the structure of the data, the transactions provided by SAP that modify that data have a much broader scope of influence. As a result, directly modifying the application database tables of an SAP application can have disastrous results to the application's data integrity.

Instead of directly modifying the database tables, SAP provides a flexible ABAP API (Call Transaction) for Create, Update, and Delete operations. Call Transaction is SAP-provided functionality for entering data into an SAP application. It guarantees that the data adheres to SAP's data model by using the same screens an online user would use in a transaction. This process is commonly referred to as screen scraping.

Retrieve processing

If the verb is Retrieve, then the connector uses ABAP SQL statements to retrieve data from the SAP application database. The business object data provides the keys for the `where` clause when pulling data. The difficulty in this methodology of retrieving data is that the retrieved data must be represented in a format that represents the business object structure. This is done in the ABAP Handler ABAP code.

The connector supports the Retrieve processing only with a primary key specified.

Return code and returned business object data

Regardless of the verb of the business object, the connector waits for two types of confirmations:

- Return code
- Returned business object data (for success only, return code = 0)

The ABAP Extension Module uses three different return codes to process business object data; return code 0, 21, and any non-zero code (other than 21). Set the return code in the function module interface. For more information on the function module interface, see "IBM WebSphere function module interface" on page 71.

Return code 0

Return code 0 indicates that the business object was successfully processed and returns VALCHANGE to the connector infrastructure. If ABAP handler processing is successful, then the connector expects new business object data that reflects the operation performed. For example, after a successful Create, the returned business object is an exact copy of the business object initially sent in, except that the keys are updated. Similarly, a successful Retrieve results in a fully formed instance of the business object. However, Create, Update, and Delete operations have different requirements for returned business objects than do Retrieve operations.

When the IBM WebSphere InterChange Server is the integration broker, the difference in requirements comes from how the WebSphere business integration system handles business objects, specifically dynamic cross-referencing of object IDs during mapping. When the connector returns a business object to IBM WebSphere InterChange Server after a Create or Update operation, the mapping infrastructure attempts to update the cross-reference tables with the newly acquired object ID. This is accomplished by looking up the value of the business object's `ObjectEventId` attribute that was set when the business object was originally sent to the connector.

To the ABAP handlers, this is significant because the ABAP handlers are responsible for "stitching" the object IDs into the business object that is returned to the connector. Typically, this is not an issue for Retrieve operations because there is no corresponding dynamic cross-referencing. Retrieve operations generate an entirely new business object that is returned to the connector. This business object does not have any direct relationship to the structure of the business object originally sent in.

The business object data returned by the ABAP handler must be in the same flat structure format as when it was initially passed in to function module `/CWLD/RFC_DO_VERB_NEXTGEN`. The ABAP handler needs to send out only simple type attributes with the following information for each:

- Value
- Peer relationship
- Application-specific information

The attribute name is not required at this point, because the connector uses only the application-specific information to create a business object from this data. Identifiers for the beginning and ending of business objects or object type attributes are not used and should not be added. For example, the `BoName` and

BoVerb rows are not used in the business object returned from the ABAP handler. They are initially passed into the ABAP handler only to facilitate processing.

The ABAP handler must adhere to the following set of rules when populating a flat structure with business object response data representing an WebSphere business object:

- Send only simple attributes, not object types.
- All attributes must exist in the WebSphere business object definition.
- All attributes must be sent in the order they are listed in the WebSphere business object definition.
- No attribute of a child business object can be sent unless at least one attribute is sent for its parent business object.
- Contained business objects must communicate the number of peers they have.
- Attribute name (field ATTR_NAME) is not required.

Figure 13 illustrates a flat business object (no object type attributes).

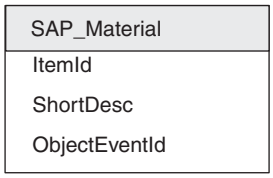


Figure 13. Flat business object SAP_Material

Table 11 represents the structure of a flat business object, SAP_Material, whose key value is ItemID. Notice that field ATTR_NAME is not required, APPTEXT is unique for each attribute, and because this business object is flat, the PEERS field can be left blank.

Table 11. Flat business object SAP_Material

ATTR_NAME	ATTR_VALUE	ISKEY	ISNEW	PEERS	OBJ_ NUMBER	APPTEXT
(blank)	000000000000001179	(blank)	(blank)	(blank)	(blank)	ItemId
(blank)	Toaster 6000	(blank)	(blank)	(blank)	(blank)	ShortDesc
(blank)	SAP_124	(blank)	(blank)	(blank)	(blank)	ObjectEventId

Figure 14 illustrates a hierarchical business object (containing object types).

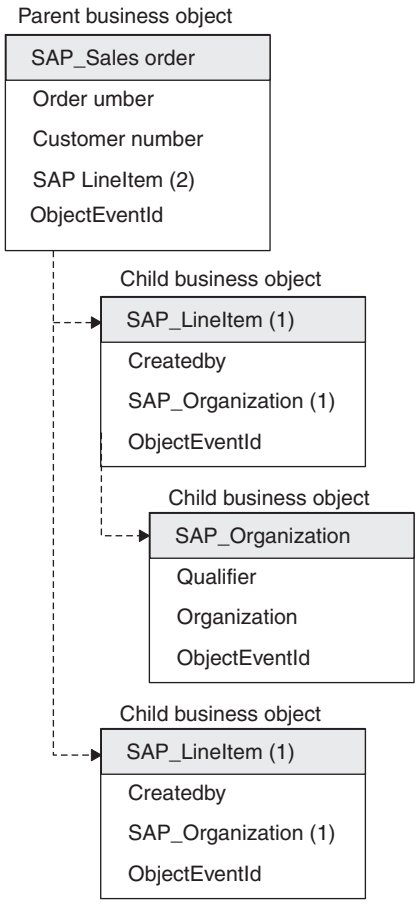


Figure 14. Hierarchical business object SAP sales order (IDoc)

Table 12 shows a representation of a flat structure of a hierarchical business object based on an IDoc Sales Order. Notice that field ATTR_NAME is not required, APPTEXT is unique for each attribute, and because this business object is hierarchical, the PEERS field lists the appropriate relationship.

Table 12. Hierarchical business object based on an IDoc sales order

ATTR_NAME	ATTR_VALUE	ISKEY	ISNEW	PEERS	OBJ_ NUMBER	APPTEXT
(blank)	USD	0	0	1	(blank)	E1EDK01:CURCY
(blank)	0000000101	0	0	1	(blank)	E1EDK01:BELNR
(blank)	SAP_124	0	0	1	(blank)	E1EDK01: ObjectEventId
(blank)	User1	0	0	2	(blank)	Z1XRV40:ERNAM
(blank)	SAP_125	0	0	2	(blank)	Z1XRV40: ObjectEventId
(blank)	001	0	0	1	(blank)	Z1XRV40:QUALF
(blank)	1000	0	0	1	(blank)	E1EDK14:ORGID
(blank)	SAP_126	0	0	1	(blank)	E1EDK14: ObjectEventId
(blank)	User1	0	0	2	(blank)	Z1XRV40:ERNAM
(blank)	SAP_127	0	0	2	(blank)	Z1XRV40: ObjectEventId

Return code 21

Return code 21 indicates that the business object was successfully processed and returns SUCCESS to the connector infrastructure. This code returns only success and does not return any business object data back to the connector. The Object-specific IDoc handler that process the business object data returns a return code of 21 when the business object data is successfully entered into the SAP application. The return code is passed back to the `/CWLD/RFC_DO_VERB_NEXTGEN` function module, which returns success back to the connector. The connector never receives business object data.

This is useful when passing large objects (such as an IDoc with multiple line items) and all you want is to know that your business object data was passed to the SAP application successfully. Performance is greatly improved because you return only the code and not the business object.

When WebSphere InterChange Server is the integration broker, you should use return code 21 only when the business object does not require cross-referencing and when you are simply passing data into the SAP application. Do not use return code 21 for retrieve operations. Behavior of the SUCCESS return code means that no business object is returned to the WebSphere InterChange Server for cross-referencing or further processing.

Non-zero return code

A non-zero return code (other than 21) indicates that the object was not processed successfully and returns FAIL to the connector. If the ABAP handler returns a non-zero code (other than 21), then no business object is returned to the connector.

Flat structure conversion to a business object

Once the flat structure has been repopulated with new business object data, `/CWLD/RFC_DO_VERB_NEXTGEN` returns the business object data to the calling connector. Remember that the connector is single-threaded; therefore, it passes only one business object at a time. The connector must now convert the business object data from the flat structure into a business object. When processing data in a flat structure into a business object, the connector must:

1. Initialize the original business object.
2. Transfer the business object data from the flat structure to the business object.
3. Deliver the business object to InterChange Server (connector controller).

Business object initialization

The connector initializes the original business object that it received from the integration broker before it populates it. When initializing the business object, the connector sets every attribute in the top-level business object to null. For object type attributes, this action recursively deletes every contained business object, leaving only the top-level business object.

How the connector rebuilds a business object

After the connector initializes the original business object, what remains is the top-level business object containing the business object name and business object verb, but no attribute value data. The attribute value data must be transferred from the flat structure from the ABAP handler. The logic for transferring the returned data is simple, but the data must be transferred in the exact order that the connector expects it.

The connector matches the application-specific information in the returned data to an attribute's application-specific information in the business object definition. The connector attempts to set every attribute that is in the returned business object data. If any attribute cannot be set, the connector returns `FAIL` to the connector infrastructure.

In order for the returned data transfer to execute successfully, the connector expects the following to be true of the returned data:

- It contains only simple attributes, where one row equals one attribute.
- Attributes must exist in the WebSphere business object definition.
- Attributes must be ordered as they are in the WebSphere business object definition (depth and then breadth).
- An attribute's application-specific information links its object's application-specific information with another value that uniquely identifies the attribute within the business object's definition.
- Child attributes must occur after their parent object's attributes (never before their parents and never after their grandparents).
- An attribute must communicate its business object's number of peers.

When the connector rebuilds the application-specific business object, the connector loops through the business object twice, beginning with the top-level business object.

1. In the first pass, it sets all simple attributes.
2. In the second pass, it checks if the flat attribute exists in a child object. If it exists the connector recursively executes the same processing for the child object.

Attention: If the conversion of a flat structure to a business object fails, the connector reports a failure to the integration broker. However, the data is already posted in the SAP application and, therefore, cannot be rolled back at this stage. While the rules are simple, implementing a complex, hierarchical business object with many attributes can be difficult to manage.

Once the business object is successfully rebuilt with new business object data, the connector returns it to the integration broker.

Chapter 6. Developing business objects for the ABAP Extension module

This chapter discusses business object development for the ABAP Extension Module. It provides background information as well as steps for developing business objects and ABAP handlers. You should be familiar with how the connector processes business objects.

This chapter contains the following sections:

Background information

Business object development for the ABAP Extension Module consists of creating an application-specific business object definition and an associated ABAP handler for each verb that you want to support.

To develop an application-specific business object, you must create a business object definition that supports your business needs. The IBM WebSphere Business Integration Adapter for mySAP.com (SAP R/3 Version 4.x) includes tools that facilitate the process of developing business object definitions in the SAP application. Although you can use Business Object Designer or a text editor to create business object definitions for the ABAP Extension Module, it is recommended that you initially use the adapter's business object development tools. These tools use the SAP application's native definitions as a template.

For each application-specific business object definition that you develop, you must support it by using an adapter-provided ABAP handler or by developing a custom ABAP handler. The ABAP handler is the mechanism that gets data into and out of the SAP application database.

Note: The application-specific business object and the ABAP handler rely on each other's consistency to pass data into and out of the SAP application. Therefore, if you change the business object definition, you must also change the ABAP handler.

An ABAP handler for the connector is implemented as an ABAP function module. ABAP handlers are one or more function modules that work together to fulfill a business object request from the business object router `/CWLD/RFC_DO_VERB_NEXTGEN`. ABAP handlers are responsible for passing business object data into and out of the SAP application.

Note: SAP supports many verbs other than those (Create, Retrieve, Update, and Delete) supported by the WebSphere business integration system. You can develop an ABAP handler to support any verb.

To develop an ABAP handler, you must understand how the connector gets data into and out of the SAP application and what form that data takes during this

process. For a high-level description of business object processing, see Chapter 3, "Overview of the ABAP Extension module," on page 33. For a detailed description of business object processing, see Chapter 5, "Business object processing in the ABAP Extension module," on page 57.

Note: When you develop business objects, you must make sure that the objects are added to the connector's table `/CWLD/OBJECTS` table in the SAP R/3 application. If they are not, then you will not be able to access the objects for customization (for example, setting up the object for event distribution).

SAP native APIs

Adapter-provided ABAP handlers use SAP native APIs, which enable ABAP handlers to pass data into and out of the SAP application. The WebSphere business integration system has implemented the following native APIs:

- "ABAP SQL"
- "Call Transaction"
- "Batch data communication (BDC)" on page 71
- "Business application programming interface (BAPI)" on page 71

ABAP SQL

ABAP SQL is SAP's proprietary version of SQL. It is database- and platform-independent, so that whatever SQL code you write, you can run it on any database and platform combination that SAP supports. ABAP SQL is similar in syntax to other versions of SQL and supports all of the basic database table commands such as update, insert, modify, select, and delete. For a complete description of ABAP SQL, its use, syntax and functionality, see your SAP documentation.

Using ABAP SQL, an ABAP handler can modify SAP database tables with business object data for create, update, and delete operations. It can also use the business object data in the `where` clause of an ABAP select statement as the keys.

Note: The WebSphere business integration system never uses ABAP SQL to modify SAP tables, because this may corrupt the integrity of the database. The connector uses ABAP SQL only to retrieve data and to modify adapter-delivered database tables.

Call Transaction

Call Transaction is SAP-provided functionality for entering data into an SAP system. Call Transaction guarantees that the data adheres to SAP's data model by using the same screens an online user sees in a transaction. This process is commonly referred to as screen scraping. To use Call Transaction, specify the following types of instructions:

- Initiation—transaction to call
- Navigation—sequence of screens to process
- Mapping—input data that should go into each field on a screen

Initiation is passed as a single value parameter in the Call Transaction call. Navigation and Mapping instructions are passed in together in a table with a specific format. This format is usable for invoking Call Transaction for any SAP transaction. In this format, these instructions are referred to as the BDC data, BDC table, or BDC session.

Batch data communication (BDC)

Batch Data Communication (BDC) is an instruction set that SAP can follow to execute a transaction without user intervention. The instructions dictate the sequence in which a transaction's screens are processed and which fields should be populated with data on which screens. All of the elements of an SAP transaction that are exposed to an online user have identifications that can be used in a BDC. The elements are as follows:

- Screens—identified by a program name and screen number
- Input fields—typically identified by the database table and field name to which it refers
- Commands in the transaction—commands such as save, new items, details, and exit (identified by a one- to eight-character code)

To get a screen's BDC identity, place the cursor in any field on the screen. Press F1 for help and then F9 for technical information. The program name and screen number are listed under Screen Data.

To get an input field's BDC identity, place the cursor in each field on the screen in which you want to input data. Press F1 for help and then F9 for technical information. If there is a box named Field Description for Batch Input, then use the information in the Screen Field field. If this box does not exist, from the Field Data box, concatenate the Table Name and Field Name together with a hyphen.

To get a command's BDC identity, highlight the command in the menu and press F1 for help. Use the value in the Function field.

Business application programming interface (BAPI)

Use the BAPI Module to support BAPIs. For more information, see Chapter 14, "Overview of the BAPI Module," on page 163.

IBM WebSphere function module interface

Every ABAP handler must implement the same function module interface. The function module interface guarantees that the business object router `/CWLD/RFC_DO_VERB_NEXTGEN` can pass business object data to and from ABAP handlers. The interface is:

```
*"*"Local interface:
*"  IMPORTING
*"        VALUE(PROC_FUNC_1) LIKE  RS38L-NAME OPTIONAL
*"        VALUE(PROC_FUNC_2) LIKE  RS38L-NAME OPTIONAL
*"        VALUE(OBJECT_NAME) LIKE  /CWLD/LOG_HEADER-OBJ_NAME OPTIONAL
*"        VALUE(OBJECT_VERB) LIKE /CWLD/WIZ_IN-OBJ_VERB OPTIONAL
*"        VALUE(ARCHIVE) OPTIONAL
*"        VALUE(TEXT) LIKE  T100-TEXT OPTIONAL
*"  EXPORTING
*"        VALUE(RETURN_TEXT) LIKE /CWLD/LOG_HEADER-OBJ_KEY
*"        VALUE(RFCRC) LIKE /CWLD/RFCRC_STRU-RFCRC
*"  TABLES
*"        RFC_STRUCTURE STRUCTURE /CWLD/OBJ_STRU
*"  EXCEPTIONS
*"        NOT_FOUND
*"        ERROR_PROCESSING
```

In the importing section of the interface, you can communicate values such as the ABAP handler name, business object name, and business object.

The exporting section of the interface is used to communicate the results of the ABAP handler processing. The return code RFCRC parameter is a single field used to determine the code a connector returns. The possible values are:

RC = 0 (success, VALCHANGE)

RC = 1 (failure, FAIL)

RC = 21 (success, SUCCESS)

The RETURN_TEXT parameter is a 120-character free text field that is written to by the connector or logged as an error message in the return status descriptor. If the ABAP handler does not provide a value for this parameter, then /CWLD/RFC_DO_VERB_NEXTGEN supplies default text depending on the return code.

Note: The exceptions section of the interface defines two exceptions. It is recommended that you use the exporting parameters instead.

IBM WebSphere ABAP handler APIs

The adapter provides several APIs that facilitate the development of ABAP handlers for the WebSphere business objects for SAP. These APIs were developed as "generic" ABAP handlers, because they require only metadata to support additional business objects of any type. The adapter provides the following ABAP handler APIs:

- Dynamic Retrieve—/CWLD/DYNAMIC_RETRIEVE
- Dynamic Transaction—/CWLD/DYNAMIC_TRANSACTION
- IDoc Handler—/CWLD/IDOC_HANDLER

The adapter provides a set of tools that support these APIs. For all three ABAP handler APIs, the tools can be found in IBM CrossWorlds Station (transaction /n/CWLD/HOME). The adapter also provides SAPODA. For more information, see Appendix E, "Generating business object definitions using SAPODA," on page 291.

The following sections discuss the adapter-provided APIs and gives you steps on how to use IBM CrossWorlds Station tools and SAPODA to develop business objects for them.

Important: You must log on to the SAP system in English when using IBM CrossWorlds Station tools to generate business object definitions or ABAP handlers. The CrossWorlds Station log is available only in English. You must also log on to the SAP system in English for the SAPODA.

Business object attribute properties

Business object architecture defines various properties for attributes. This section describes how the connector interprets several of these properties and describes how to set them when modifying a business object. Table 13 lists the business object attribute properties for the ABAP Extension Module.

Table 13. Business object attribute properties for the ABAP Extension Module

Property name	Description
Name	Each business object attribute must have a unique name.
Type	The value is String.
MaxLength	This property is not used.

Table 13. Business object attribute properties for the ABAP Extension Module (continued)

Property name	Description
IsKey	The first simple attribute of a business object is set as the key attribute. All key attributes should be of type String. Setting a child object as a key attribute is not supported.
IsForeignKey	This property is not used.
IsRequired	This property specifies whether an attribute must contain a value.
AppSpecificInfo	The value of this property is different depending on which ABAP handler supports the business object. The adapter delivers business object generation tools that automatically provide this value. If you modify the generated value, the business object may fail to process properly.
DefaultValue	This property specifies the value to assign to this attribute if there is no run-time value.

Adapter development tools

The adapter provides business object development tools that let you generate a WebSphere business object definition file from within the SAP application. This business object definition file directly corresponds to the SAP business process and API from which it was generated.

Note: When the IBM WebSphere InterChange Server is the integration broker, verify that your final business object definition file contains the version at the top. Early versions of the WebSphere InterChange Server require version text, which is located in the `\repository\ReposVersion.txt` file under the product directory. Also verify that the definition includes all of the required business objects and attributes (including the `ObjectEventID` attribute).

In IBM CrossWorlds Station, the following development tools are available:

- Advanced Outbound Wizard
- Inbound Wizard
- Object Definition Generator

Important: You must log on to the SAP system in English when using IBM CrossWorlds Station to generate business object definitions or ABAP handlers. The CrossWorlds Station log is available only in English. You must also log on to the SAP system in English for the SAPODA.

Note: For information on using Advanced Outbound Wizard, which creates hierarchical or flat business object definitions by stepping through the desired SAP transaction, see "Generating business objects: Advanced Outbound Wizard" on page 225.

Inbound wizard

The Inbound Wizard tool enables you to define business objects and the metadata required for their processing by recording your actions as you step through an SAP transaction that supports your required functionality. You do not need to write any ABAP code nor do you need to know the underlying database schema for the business object.

The Inbound Wizard generates the data for the Dynamic Transaction table by recording and interpreting user actions in an SAP transaction. It supports the definition of flat (non-hierarchical) business objects. In other words, it does not

support business objects that contain child business objects. The Inbound Wizard can be used as a code generator to facilitate the development of more complex objects that require static code.

Note: You can manually develop new business objects or modify existing business objects by adding/modifying entries to the Dynamic Transaction table.

For more information on developing business objects for business object requests, see "Developing business objects using dynamic transaction."

Developing business object definitions using object definition generator

The Object Definition Generator enables you to build a WebSphere business object definition based on an IDoc or on the metadata in the Dynamic Transaction table. The business object definition file that is produced maintains the relationships and structure of the IDoc. The IDoc handler uses business objects developed from these IDocs. Therefore, the generator allows you to add your object-specific IDoc handler function modules when you generate the business object definition.

Once the business object definition is generated, you need only modify attribute names and make sure that the definition supports all of the desired functionality.

Note: This tool is used primarily to generate business objects based on IDocs, but can also be used for Dynamic Transaction.

For more information on developing business objects using the Object Definition Generator, see "Developing business objects using IDocs" on page 79.

Developing business object definitions using SAPODA

SAPODA enables you to build a WebSphere business object definition based on an IDoc, or the tables used by Dynamic Retrieve and Dynamic Transaction. For more information on developing business objects using SAPODA, see Appendix E, "Generating business object definitions using SAPODA," on page 291.

Developing business objects using dynamic transaction

The Dynamic Transaction function module is a mapping tool and dynamic code generator. It uses SAP's Call Transaction API to get data into an SAP application. Also, it stores static definitions of Batch Data Communication (BDC) sessions by combinations of object and verb. Before the BDC data is passed to a Call Transaction, the business object attribute values are mapped into the BDC session. At the completion of the call transaction, the resulting key value is set in the appropriate value of the business object, and all messages from the call transaction are logged.

The Dynamic Transaction function module builds a BDC session to do a call transaction by combining the BDC defined in the Dynamic Transaction table, `/CWLD/WIZ_IN`, and the values from the incoming business object. When the Dynamic Transaction function module is called, the following steps are performed:

1. All entries are retrieved from `/CWLD/WIZ_IN`, where:

   `object name = objectName and verb = objectVerb`
2. Field input values are mapped from the business object into the BDC session based on the attribute name.
3. BDC sessions are processed using Call Transaction.

4. Key values are captured, Call Transaction messages are logged, and the key is set in the business object.

Tips

- Data entered on an initial screen may default for all line items and reduce required line item input.
- Line item overview screens may provide enough input rather than drilling down to a details screen, which may require additional input.
- Confirmation messages usually do not need to be answered in BDC; for example, `Are you sure you want to save?`
- The counter renumbers in increments of 10, for each object and verb combination, every time you enter and exit the table maintenance in change mode.
- During execution, the Call Transaction uses the user's settings for date formatting. Be sure the connector's user is set up to use a variation of `YYYY-MM-DD` date format, because this is the standard date format used by the WebSphere business integration system. Similarly, change your own user settings if you want to reprocess the business objects by stepping through the transaction.

Composing a BDC session for a business object

Composing a BDC session requires an understanding of an SAP transaction's design. An SAP transaction allows the same data to be input in various sequences and on different screens. Typically each sequence or flow exposes additional functionality. As a result, certain data validation and input field requirements occur on some screens, but not on others. The challenge is to find the sequence that does what you need with the least amount of effort. A simple BDC session is more stable than a complex BDC session.

An SAP transaction may behave differently when accessed using the Call Transaction method in a background process instead of executing online. For example, different or additional screens may appear or input fields may reside on screens different from those your online investigation revealed. The discrepancy occurs because the transaction's controlling code may dictate different behavior when executed in the background from that executed online. As a result, your online test may work when reprocessing a failed object event as you step through the transaction; however, the connector consistently fails when processing the same object. If this occurs, modify the BDC so that it processes in the background. If you modify the BDC, you may encounter cases where the BDC processes in the background, but fails when processed online.

The BDC you define in the Dynamic Transaction table is static. It cannot react during the transaction if certain input data causes other screens to pop up or other fields become mandatory during runtime. Proper investigation of a transaction's configuration is important to be able to predict consistent behavior. Experiment several times with the transaction; repeated behavior can become your guideline.

Once you have determined the screen flow, follow the steps below and document the information you gather in a spreadsheet.

1. Go to the transaction that supports your object and identify the transaction code.
2. Identify the BDC elements for the screen and input fields you require.

3. Identify the menu command you need to continue processing to the next screen.
4. Repeat steps 2 and 3 for each screen required.
5. Conclude by noting the command to save the transaction.

Table 14 describes the column names for Dynamic Transaction table /CWLD/WIZ_IN.

Table 14. /CWLD/WIZ_IN table entries for dynamic retrieve

Field name	Description	When used	Technical name
Object name	IBM WebSphere business object name	Always	OBJ_NAME
Verb	Verb (Create, Update, Delete, or Retrieve)	Always	OBJ_VERB
Counter	Counter	Always	POSNR
Program	Program associated with a screen	BDC screen identification	PROG_NAME
Screen number	Screen number associated with a screen	BDC screen identification	DYNPRO
Start	Specifies a new screen	BDC screen identification	DYNBEGIN
Screen description	Free text description of screen, field, or command	User discretion	SCR_DESCR
BDC field name	BDC input field name	BDC input fields	FNAM
Field name in business object	Attribute in the IBM WebSphere business object to supply the input value	BDC input fields	SOURCEFLD
Default value	A static default value to use if no entry is provided in the IBM WebSphere business object or if using BDC_OKCODE, because it is the command value	A value might not always be passed in and it is mandatory for the transaction	DEFLT_VAL
SY Field name	A dynamic system field to be used as a default value (for example: DATUM)	A value is not passed in or should be determined by SAP system fields	SYFIELD
Return	A number between 1 and 4, that identifies which system message field returns the key value at transaction completion (SY-MSGV#)	A business object key attribute that should receive the key value	RETURNFLD
Length	Character length from the zero position of the attribute value that should be used for input	Relevant only when using an attribute that contains a composite value	LENGTH

To define or modify a business object's metadata (transferring information to /CWLD/WIZ_IN):

1. Go to IBM CrossWorlds Station (transaction /n/CWLD/HOME).

   Important: You must log on to the SAP system in English when using IBM CrossWorlds Station to generate business object definitions or ABAP handlers. The CrossWorlds Station log is available only in English.
2. Click the Development tab.

3. Click the Modify BO Metadata button in the Transaction based - Inbound section.

Defining the metadata for the business object is simple. For each screen, the first entry identifies the screen, subsequent entries identify the input fields, and the last entry must be a command. This grouping repeats for each screen.

Using the Counter column as a line number for discussion, step through the SAP4_CustomerMaster example.

100 Begin with screen number 100 of program SAPMF02D. This is a new screen, the first, so it is flagged in the Start column.

110 On screen 110, use the value from the Customer_account_group attribute in the business object, and add it to the BDC field name column (the value is RF02D-KT0KD). Specify the default value as 0001. If the Customer_account_group attribute contains CxIgnore, then the BDC field name column receives the default value 0001

120 The Customer_Account_Number attribute is the key value, so it is not set during the Call Transaction. SAP assigns the key value internally and makes it available only after the transaction is successfully posted. For this reason, leave the BDC field name column blank, but include an entry in the table because the Customer_Account_Number attribute must be set with this key value when it is returned at the conclusion of the Call Transaction. Also enter the word RETURN in the Program column for CustomerNumber.

Depending on the transaction, SAP returns the key value in one of four possible fields: SY-MSGV1, SY-MSGV2, SY-MSGV3 or SY-MSGV4. To specify that you want the return value set in a particular attribute, enter a number, 1-4, in the Return column. This number corresponds to the SY-MSGV# field containing the key value.

130 You are finished entering the necessary values for the first screen, so enter a command, /00, in the Default Value column to simulate pressing the Enter key. This takes you to the next transaction screen. Commands are entered in the screen input field, BDC_OKCODE, which is where you enter in a transaction code.

140 At this point, you are at the next transaction screen. Enter the address information. Since it is a new screen, flag it in the Start column. In this example, the second screen is associated with the same program as the initial screen, and only the screen number changed from 100 to 110. This is not always the case.

150 -210 Use the values from the Name_1, Sort_field, City, P_0_Box_postal_code, Country_key, Language_keys, and Post_office_box attributes in the business object, and add corresponding values to the BDC field name column.

220 Similar to line 130, processing for this screen is complete. However, rather than simply simulating the Enter key, enter the command value UPDA to save the transaction. This takes you to the next transaction screen.

230 At this point, you are at the third transaction screen, so flag it in

the Start column. Because your business object does not require data from this screen, you will complete processing for this screen in the next line.

240 Similar to line 130, processing for this screen is complete. Enter the command value `/00` to simulate pressing the Enter key. This takes you to the final transaction screen.

250 At this point, you are at the final transaction screen. Flag it in the Start column.

260 Similar to lines 150-210. Use the value from the business object attribute, *Transport_zone_to_which_or_from_which_the_goods_are_delivered*, and add its corresponding value (`KNA1-LZONE`) to the BDC field name column.

270 Similar to line 220, processing for this screen is complete and the transaction is complete, so enter the command value to save, `UPDA`. This is the last action the Call Transaction API receives.

280 The final entry for any business object is always the specification of the transaction code. The keyword `TCODE` goes in the Program column and the transaction code goes in the BDC field name column.

This completes the definition of the BDC Session for the SAP4_CustomerMaster business object.

If a call transaction returns an error message when it fails, you could have one of the common errors described below.

- The SAP application has called a screen that the BDC did not expect, so the SAP application returns the message, `No input available for program XX and screen YY`. If this occurs, add the appropriate entries to the Dynamic Transaction table to handle the input screen for program XX and screen YY.
- The SAP application is instructed by the BDC to set a field that does not exist. Most likely, the SAP application executed its own instruction that you did not explicitly set. As a result, you are on a different screen than you intended. If this occurs, repeat the instruction and add only the piece that sends you to the appropriate screen.

Using the inbound wizard for dynamic transaction

The Inbound Wizard records your navigation, actions, and field inputs in a transaction when you click the first field or change screens. The recorder picks up every action that occurs, but not everything you see. For example, when the initial screen first appears, the recorder captures the initial call to the transaction, but not all of the input fields that appear on that screen. If you want to be able to use an input field, you must enter some data into that field. Also, even though an input field may contain default data, that data is not picked up unless it is manually entered.

To create a new WebSphere business object definition:

1. Go to IBM CrossWorlds Station (transaction `/n/CWLD/HOME`).

Important: You must log on to the SAP system in English when using IBM CrossWorlds Station to generate business object definitions or ABAP handlers. The CrossWorlds Station log is available only in English.

2. On the Development tab, click the Inbound Wizard Button.
3. Enter the following information:
   - Business Object Name—Name of the business object type as well as the name of every instance of the object. If you are creating a new business object, then enter a new name. It is recommended that you use a simple name that defines the business object. If you are using an existing business object, then select it from the drop-down list.
   - Verb—Verb supported by the business object.
   - Transaction Code—Transaction code for the screen that supports the necessary functionality performed by the business object. To get the transaction code for a screen, Click Status from the System menu. The code is listed in the Transaction field under SAP data.
4. Click Record.
5. Step through the transaction that supports your business object functionality. Use all necessary fields and screens. When you are finished, save your transaction.
6. Choose the components that you want to include as metadata in your business object. Place your cursor on the component, and then click the Select/Deselect sub-tree button (F9). By default, all components are selected.
7. Generate a new dynamic object or source code.
   - To generate and insert the metadata for the Dynamic Transaction table, click the Generate Meta data button (F6). You can generate a WebSphere business object definition from this data.
   - To generate a text file with BDC data and field descriptions, click the Generate Code in Text File button (F5). You cannot generate a WebSphere business object definition from this data.

Developing business objects using IDocs

WebSphere business objects for the ABAP Extension Module can be defined in SAP as an IDoc. IDocs are part of SAP's EDI solution known as ALE (Application Link Enabling). Their definitions are stored in SAP's BOR (Business Object Repository) and can be accessed globally within an SAP system. This IBM WebSphere Business Integration Adapter for mySAP.com leverages the definition part of ALE to interpret and parse WebSphere business objects in the SAP application in preparation for use with an SAP native API. The adapter provides an IDoc handler that supports business objects developed using IDocs.

IDoc handler consists of two function modules. Other ABAP handlers such as Dynamic Retrieve and Dynamic Transaction consist of only a single function module.

`/CWLD/RFC_DO_VERB_NEXTGEN` passes business object data to IDoc handler `/CWLD/IDOC_HANDLER`. This IDoc handler, which is generic to all object types, uses the business object's application-specific information to obtain the type of IDoc specified and to reformat the business object data into the structure of that IDoc. After it reformats the data, the generic IDoc handler passes the business object data to an object-specific IDoc handler (based on the combination of the business object type and its verb), which handles the integration with an SAP native API. After the

object-specific IDoc handler finishes processing the business object data, it returns the business object data in IDoc format to /CWLD/IDOC_HANDLER. This generic IDoc handler now converts the business object data back to its original format and returns it to /CWLD/RFC_DO_VERB_NEXTGEN.

Figure 15 illustrates the basic architecture of IDoc handler.

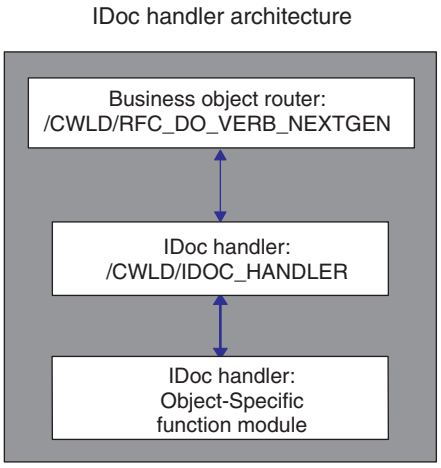


Figure 15. IDoc h'andler architecture

To use the adapter-provided IDoc handler, you must have an IDoc defined in the SAP application. Either SAP-delivered or customer-built IDocs can be used. Because an IDoc definition must mirror the definition of the WebSphere business object for SAP, the adapter provides two tools to generate the WebSphere business object definition based on an IDoc:

- A tool in IBM CrossWorlds Station
- SAPODA

The following sections describe how to use both of these tools.

Using IBM CrossWorlds Station to generate a business object definition

Before you can use IBM CrossWorlds station to generate a business object definition, you must have already created a WebSphere business object in the SAP application. To create a business object definition based on an IDoc:

1. Go to IBM CrossWorlds Station (transaction /n/CWLD/HOME).

   Important: You must log on to the SAP system in English when using IBM CrossWorlds Station to generate business object definitions or ABAP handlers. The CrossWorlds Station log is available only in English.
2. Click the Tools tab.
3. Click the CW Object Definition button, and then click Advanced Download (F7).
4. Fill in the Object and Functions information as needed. You must specify an existing IDoc type and WebSphere Object Name. The fields in the Functions section are the application-specific information for the supported verbs.

For example, an SAP4_Order business object based on the Order IDoc type YXRV4B01 could have the following functions:

- Create: /CWLD/IDOC/HANDLER:Y_XR_ORDER_C1
- Update: /CWLD/IDOC/HANDLER:Y_XR_ORDER_C2
- Delete: /CWLD/IDOC/HANDLER:Y_XR_ORDER_C3
- Retrieve: /CWLD/IDOC/HANDLER:Y_XR_ORDER_C4

For more information on how the application-specific information is used for the verb functions, see "Business object data routing to ABAP handlers" on page 61.

After defining the IDoc, create a function module for each verb the business object must support. Each function should have the following interface to ensure that /CWLD/IDOC_HANDLER can call it:

```
*" IMPORTING
*"        VALUE(OBJECT_KEY_IN) LIKE /CWLD/LOG_HEADER-OBJ_KEY OPTIONAL
*"        VALUE(INPUT_METHOD) LIKE  BDWFAP_PAR- NPUTMETHD OPTIONAL
*"        VALUE(LOG_NUMBER) LIKE /CWLD/LOG_HEADER-LOG_NR OPTIONAL
*" EXPORTING
*"        VALUE(OBJECT_KEY_OUT) LIKE /CWLD/LOG_HEADER-OBJ_KEY
*"        VALUE(RETURN_CODE) LIKE /CWLD/RFCRC_STRU-RFCRC
*"        VALUE(RETURN_TEXT) LIKE /CWLD/LOG_HEADER-OBJ_KEY
*" TABLES
*"        IDOC_DATA STRUCTURE  EDID4
```

Using SAPODA to generate a business object definition

You can use SAPODA to generate business object definitions for the ABAP Extension Module based upon an IDoc:

- Extracted as a file
- Defined in the SAP System

Important: You must log on to the SAP system in English to use SAPODA.

When you use SAPODA to generate a business object definition, you can use Business Object Designer to view and modify the definition. For more information on using SAPODA, see Appendix E, "Generating business object definitions using SAPODA," on page 291.

After defining the IDoc, create a function module for each verb the business object must support. Each function should have the following interface to ensure that /CWLD/IDOC_HANDLER can call it:

```
*" IMPORTING
*"        VALUE(OBJECT_KEY_IN) LIKE /CWLD/LOG_HEADER-OBJ_KEY OPTIONAL
*"        VALUE(INPUT_METHOD) LIKE  BDWFAP_PAR- NPUTMETHD OPTIONAL
*"        VALUE(LOG_NUMBER) LIKE /CWLD/LOG_HEADER-LOG_NR OPTIONAL
*" EXPORTING
*"        VALUE(OBJECT_KEY_OUT) LIKE /CWLD/LOG_HEADER-OBJ_KEY
*"        VALUE(RETURN_CODE) LIKE /CWLD/RFCRC_STRU-RFCRC
*"        VALUE(RETURN_TEXT) LIKE /CWLD/LOG_HEADER-OBJ_KEY
*" TABLES
*"        IDOC_DATA STRUCTURE  EDID4
```

IDoc handlers and create, update, and delete verbs

IDoc handlers that support Create, Update, and Delete operations receive business object data formatted as an IDoc. The role of these operations is to integrate the business object data with SAP's Call Transaction API and generate an object key. Only the object key is passed back through /CWLD/IDOC_HANDLER to the connector,

not the business object data. /CWLD/IDOC_HANDLER stores the business object data in memory and inserts the object key into the first attribute marked IsKey in the parent business object. Then, /CWLD/IDOC_HANDLER passes the business object data back to the connector.

Note: When the WebShpere InterChange Server is the integration broker, it is critical to maintain the business object data because the mapping infrastructure requires the preservation of the ObjectEventId for dynamic cross-referencing.

The sample code below represents the following flow:

1. Initializes global data.
2. Deconstructs the IDoc into working tables.
   - Initializes the target structure with '/' (CxIgnore) because not all objects are sent into the SAP application. Uses the form in /CWLD/INBIDOC_FRMS0.
   - Uses the forms in /CWLD/INBIDOC_FRMS0 to transfer data from the IDoc into internal tables for consistent behavior across objects.
3. Builds the BDC. Uses the forms in /CWLD/INBIDOC_FRMS0 to transfer data from the internal tables into the BDC table for consistent behavior across objects.
4. Creates a Call Transaction.
5. Captures the object key.

The following sample code supports SAP Sales Quote Create:

```
*- Initialize working variables and internal tables
  PERFORM INITIALIZE_IN.

*- I01(MF): Begin IDoc interpretation
  PERFORM LOG_UPDATE(/CWLD/SAPLLOG) USING C_INFORMATION_LOG TEXT-I01
                                     SPACE SPACE SPACE.

*- Interpret IDoc data structure
  IF NOT IDOC_DATA[] IS INITIAL.

*- Move IDoc to internal tables
    PERFORM INTERPRET_IDOC.

*- Check some of the input fields
    PERFORM CHECK_INPUT.

*- If key values were missing, exit function
    IF RETURN_CODE NE 0.
      EXIT.
    ENDIF.

*- E01(MF): No Idoc data lines sent for processing.
  ELSE.

    RETURN_CODE = 2.
    RETURN_TEXT = TEXT-E01.
    EXIT.

  ENDIF.

*- Build the BDC session for transaction VA21.
  PERFORM BUILD_BDC_VA21.

*- Call Transaction
  PERFORM LOG_UPDATE(/CWLD/SAPLLOG) USING C_INFORMATION_LOG TEXT-I02
                                     'VA21' C_BLANK C_BLANK.
```

```
  CALL TRANSACTION 'VA21' USING BDCDATA
                           MODE INPUT_METHOD
                         UPDATE 'S'
                       MESSAGES INTO BDC_MESSAGES.

*- Capture return code and object key from transaction
  PERFORM PREPARE_RETURNED_MESSAGE.

ENDFUNCTION.
```

The Create logic has two main functions:

- Translates the IDoc Data into manageable data structures
- Executes a Call Transaction

Translating IDOC structure

The first part of the Create logic is the task of translating data in the IDoc structure into working data structures. To do this, you need to create code similar to the following:

```
loop at idoc_data.

    case idoc_data-segnam.
      when 'ZSQVBAK'.                 " Header Data
        move idoc_data-sdata to zsqvbak.

      when 'ZSQVBUK'.                 " Status Segment
        move idoc_data-sdata to zsqvbuk.

      when 'ZSQVBP0'.                 " Partner Header Level
        move idoc_data-sdata to zsqvbp0.

      when 'ZSQVBAP'.                 " Item Detail
        move idoc_data-sdata to zsqvbap.

      when 'ZSQVBA2'.                 " Item Detail Part 2
        move idoc_data-sdata to zsqvba2.

      when 'ZSQVBUP'.                 " Item Status
        move idoc_data-sdata to zsqvbup.

      when 'ZSQVBKD'.                 " Commercial data
        move idoc_data-sdata to zsqvbkd.

      when 'ZSQKONV'.                 " Condition
        move idoc_data-sdata to zsqkonv.

      when 'ZSQVBPA'.                 " Partner Item Level
        move idoc_data-sdata to zsqvbpa.

    endcase.

  endloop.
```

Creating inbound call transaction logic

The second part of the create logic performs the operation of adding data to the SAP application database. You can use available functionality such as BAPIs and SAP standard functions or you can use custom-developed Call Transaction functionality. Keep in mind that if you use the available functionality, it may change in future releases. It is recommended that you use Call Transactions instead of writing to the database. Call Transactions allow you to develop custom functionality, independent of SAP database changes and specific to the scope and functionality you need.

To pass business object data into SAP, you can generate some of the ABAP code by using the Inbound Wizard in IBM CrossWorlds Station (transaction /n/CWLD/HOME), using the SAP BDC recorder, or developing it manually.

The Inbound Wizard records the activity for your create transaction and creates a text file with the BDC logic. For the Sales Quote example, transaction VA21 is recorded.

To record transaction VA21 using Inbound Wizard:

1. Go to IBM CrossWorlds Station (transaction /n/CWLD/HOME).

   Important: You must log on to the SAP system in English when using IBM CrossWorlds Station to generate business object definitions or ABAP handlers. The CrossWorlds Station log is available only in English.
2. On the Development tab, click the Inbound Wizard button.
3. Enter the following information:
   - Business object name—Name of the business object type as well as the name of every instance of the object. If you are creating a new business object, then enter a new name. It is recommended that you use a simple name that defines the business object. If you are using an existing business object, then select it from the drop-down list.
   - Verb—Verb supported by the business object.
   - Transaction code—Transaction code for the screen that supports the necessary functionality performed by the business object. To get the transaction code for a screen, Click Status from the System menu. The code is listed in the Transaction field under SAP data.
4. Click Record.
5. Step through the transaction that supports your business object functionality. Use all necessary fields and screens. When you are finished, save your transaction.
6. Choose the components that you want to include as metadata in your business object. Place your cursor on the component, and then click the Select/Deselect sub-tree button (F9). By default, all components are selected.
7. Generate a new dynamic object or source code.
   - To generate and insert the metadata for the Dynamic Transaction table, click the Generate Meta data button (F6). You can generate a WebSphere business object definition from this data.
   - To generate a text file with BDC data and field descriptions, click the Generate Code in Text File button (F5). You cannot generate a WebSphere business object definition from this data.

The following sample code is an excerpt from the first few lines of the generated BDC session:

```
* Sales doc. Initial screen Create
perform dynpro_new using 'SAPMV45A' '0101' .

* Sales document type
perform dynpro_set using 'VBAK-AUART' 'QT' .

* Distribution channel
perform dynpro_set using 'VBAK-VTWEG' 'sourcefield' .

* Division
```

```
perform dynpro_set using 'VBAK-SPART' 'sourcefield' .

* Function Command
perform dynpro_set using 'BDC_OKCODE' '=ENT2' .

* 4.0: Screen Container for Overview Screens (normal header)
perform dynpro_new using 'SAPMV45A' '4001' .

* Sold-to party
perform dynpro_set using 'KUAGV-KUNNR' '238' .

* Ship-to party
perform dynpro_set using 'KUWEV-KUNNR' '238' .

* Function Command
perform dynpro_set using 'BDC_OKCODE' '=KKAU' .

* 4.0: Screen container for document header screens
perform dynpro_new using 'SAPMV45A' '4002' .

* Date until which bid/quotation is binding (valid-to date)
perform dynpro_set using 'VBAK-BNDDT' '20000630' .
```

You can also use SAP's BDC recorder (transaction SHDB). The following sample code was generated using the BDC recorder:

```
start-of-selection.

read dataset dataset into record.
 if sy-subrc <> 0. exit. endif.

 perform bdc_dynpro      using 'SAPMV45A' '0101'.
 perform bdc_field       using 'BDC_CURSOR'
                               'VBAK-AUART'.
 perform bdc_field       using 'BDC_OKCODE'
                               '=ENT2'.
 perform bdc_dynpro      using 'SAPMV45A' '4001'.
 perform bdc_field       using 'BDC_OKCODE'
                               '=KKAU'.
 perform bdc_field       using 'BDC_CURSOR'
                               'KUWEV-KUNNR'.
 perform bdc_field       using 'KUAGV-KUNNR'
                              record-KUNNR_001.
 perform bdc_field       using 'KUWEV-KUNNR'
```

The output from this method does not have the business object comments from the first method and is less preferable. The advantage using SAP's BDC recorder is that it produces an independent method to verify your recording of BDC.

Another method is to generate the BDC manually. This is not an advised approach for the entire create functionality but rather as a supplement to the previous methods. It is useful when you need to add logic for additional screens or pop up boxes that may occur if the input data causes the SAP transaction to produce them.

IDoc handlers and the retrieve verbs

Object-specific IDoc handlers that support the Retrieve verb do not receive business object data from `/CWLD/IDOC_HANDLER`. Instead `/CWLD/IDOC_HANDLER` uses the `OBJECT_KEY_IN` parameter of the object-specific IDoc handler function to pass only the value of the first attribute marked `IsKey`. It is the object-specific IDoc handler's responsibility to use the value of this attribute to retrieve all information relevant to the instance of the business object using ABAP SQL, and to format that data in the appropriate IDoc structure.

Note: In cases where the key is composed of multiple fields, the event detection mechanism (or, when the WebSphere InterChange Server is the integration broker, the map) concatenates the values of these fields into the first key attribute of the top-level business object. /CWLD/IDOC_HANDLER takes this concatenated key and loads it into its OBJECT_KEY_IN parameter. The object-specific IDoc handler must parse the value of the OBJECT_KEY_IN parameter into the multiple key fields. To maintain this functionality, it is important that you do not specify name-value pairs for the key when using /CWLD/IDOC_HANDLER.

The code fragment below illustrates an object-specific IDoc handler for retrieval of a Sales Quote. The Sales Quote business object retrieves data from tables VBAK, VBUK, VBPO, VBAP, VBUP, VBKD, KNOV, and VBPA. The tables follow the hierarchy and cardinality of IDoc type ZSLSQUOT. The code does the following:

1. Initializes global data.
2. Returns business object data from the SAP application database.
3. Builds an IDoc from the returned data and returns that data to /CWLD/IDOC_HANDLER.

The code fragment for an object-specific IDoc handler for IDoc type ZSLSQUOT is:

```
*- Clear the interface structures.
  clear: g_text, object_key_out, return_code, return_text, idoc_data.
  refresh: idoc_data.

*  If no key value is specified, log it as an error and exit.
  if object_key_in is initial or
     object_key_in = c_cxignore_const.
    perform log_update(/cwld/sapllog) using c_error_log text-e02
                                       space space space.
    return_code = 1.
    return_text = text-e02.
    exit.
  endif.

  perform initialize_global_structures.

  perform fill_internal_tables.
  if not return_code is initial.
    exit.
  endif.

* Build Idoc segments from internal tables
  perform fill_idoc_inttab.

  return_code = 0.
  return_text = text-s01.

  perform log_update(/cwld/sapllog) using c_information_log text-s01
                                     space space space.
endfunction.
```

The two most important parameters are OBJECT_KEY_IN for the inbound key and IDOC_DATA for the outbound data. Note that OBJECT_KEY_IN may be a concatenated string that represents a multiple key (depending on the conventions you have defined). The object-specific IDoc handler parses the concatenated value and loads its parts into the appropriate key fields. To maintain this functionality, it is important that you do not specify name-value pairs for the key when using /CWLD/IDOC_HANDLER.

The VBAK table drives the selection criteria for the child tables, so each table is loaded into working tables. Using the VBAK table, you can retrieve the child tables with additional keys. So, for the Sales Quote example, the code is as follows:

```
form fill_internal_tables.

  * Get information from VBAK, VBUK, VBAP, VBKD, KONV, VBPA

  select single * from vbak
         where vbeln = object_key_in.

  if sy-subrc <> 0.
    perform log_update(/cwld/sapllog) using c_error_log text-e01
                                object_key_out c_blank c_blank.
    return_code = '1'.
    g_text = text-e01.
    replace '&' with order_number into g_text.
    return_text = g_text.

    exit.
  endif.

  select single * from vbuk
         where vbeln = vbak-vbeln.

  select * from vbap into table t_vbap
         where vbeln = vbak-vbeln.

  * Continue for other tables
```

The following code is used to copy the requested data from the application database into internal tables and working variables. Then it creates segments that directly correspond to the WebSphere business object definition and puts them into the SAP segment structure.

In some cases for close matches on fields between the IDoc type and the working structure, you can do an ABAP move-corresponding command. In other cases, it is preferable to manually move fields from the working table to the IDoc type table because of the relatively few fields to move in comparison to the overall number of fields in the structure. Simply, it is used to transfer data from the working data structures into the IDoc structures and then into the flat data field.

The code is:

```
form fill_idoc_inttab.

perform fill_zsqvbak.                   " Fill the Sales Quote Header
perform fill_zsqvbuk.                   " Fill the Sales Quote Status
perform fill_zsqvbap.                   " Fill Sales Quote Lines

endform.                                " FILL_IDOC_INTTAB

*-- fill the Sales Quote Header
form fill_zsqvbak.

  clear idoc_data.
  clear zsqvbak.
  idoc_data-segnam = 'ZSQVBAK'.

  move-corresponding vbak to zsqvbak.
  move zsqvbak to idoc_data-sdata.
  append idoc_data.

endform.                                " FILL_ZSQVBAK
```

```
*-- fill the Sales Quote Header Status
form fill_zsqvbuk.

  clear idoc_data.
  clear zsqvbuk.
  idoc_data-segnam = 'ZSQVBUK'.

  move-corresponding vbuk to zsqvbuk.
  move zsqvbuk to idoc_data-sdata.
  append idoc_data.

endform.                                " FILL_ZSQVBAK

*-- fill the Sales Quote Line and the Line Child segments
form fill_zsqvbap.

  loop at t_vbap.
    clear idoc_data.
    clear zsqvbap.
    idoc_data-segnam = 'ZSQVBAP'.

    move-corresponding t_vbap to zsqvbap.
    move zsqvbap to idoc_data-sdata.
    append idoc_data.

    perform fill_zsqvba2.
    perform fill_zsqvbup.
    perform fill_zsqvbkd.
    perform fill_zsqkonv.
    perform fill_zsqvbpa.


  endloop.

endform.

*-- fill second part of vbap
form fill_zsqvba2.
" etc.
```

Calling the ABAP Extension Module and ABAP handler

The connector uses the value of the verb application-specific information in a business object to call the appropriate ABAP handler in the ABAP Extension Module. To call the appropriate ABAP handler in the ABAP Extension Module, you can specify the classname for the ABAP Extension Module and must specify the ABAP handler function module used by the business object. For example, the verb application-specific information for the Dynamic Transaction ABAP handler supporting SAP R/3 version 4.6 is:

```
AppSpecificInfo = sap.sapextensionmodule.VSapBoHandler,:/CWLD/DYNAMIC_TRANSACTION
```

Note: You must uses a comma delimiter between the connector module (classname) and ABAP handler.

For more information on business object processing for the ABAP Extension Module, see “Business object processing” on page 36.

Chapter 7. Developing event detection for the ABAP Extension module

Event detection is part of the event triggering process in the ABAP component of the ABAP Extension Module. Every event detection mechanism must call an event trigger, which takes the detected event and adds it to an event table. For more information on triggering events, see "Event triggering" on page 42.

This chapter contains the following sections:

Designing an event detection mechanism

You can use many different mechanisms to detect events in the SAP application. An event detection mechanism should have the ability to make a function module call. The four event detection mechanisms that the IBM WebSphere Business Integration Adapter for mySAP.com has implemented are:

- Code enhancement—implemented for a business process (normally a single SAP transaction) by inserting event detection code at an appropriate point within the SAP transaction
- Batch program—involves developing an ABAP program containing the criteria for detecting an event
- Business workflow—uses SAP's own object-oriented event detection capabilities
- Change pointer—implementing a change pointer mechanism, which is a variation of business workflow, uses the concept of change documents to detect changes for a business process

It is important that you determine the appropriate event detection mechanism to implement for each business object that you develop, because some may not be available for a particular business process. Technical and functional knowledge of a particular business process is necessary for each transaction for which you want to implement event detection.

Review the following implementation considerations when determining which event detection mechanism to implement for your business process.

Availability	Which event detection mechanisms are available for this business process? This should be one of the first questions that you consider. Code enhancement and batch program have high availability, whereas business workflow and change pointer do not.
Real-time integration	Do events need to be detected synchronously? Do you need to detect a large number of events at one time? All mechanisms except batch program are suitable for real-time integration.
Reliability	Are all data changes for this business process detected when generating an event? Code Enhancement, Batch Program, and Change Pointer

provide the best control of capturing all events of an object. Business Workflow provides limited reliability. For example, Business Workflow does not detect an address change during a Vendor transaction update.

Flexibility — Do certain criteria need to be evaluated before an event is triggered? Does an event need to be detected at a certain point in the transaction? Code Enhancement is the most flexible, because you can insert code at a specific point before event data is committed. Change Pointer and Batch Program are moderately flexible, while Business Workflow has very little flexibility in its implementation.

Upgrade dependency — Does an upgrade to the SAP application change the way an event is detected for this business process? Typically, this is not known, but business workflow and change pointer are affected by application changes the most because they are under SAP's control.

Difficulty — Is time or level of difficulty an issue? Each mechanism has its own level of implementation difficulty. In general, batch program is the easiest. code enhancement and business workflow are moderately more difficult, while change pointer is the most difficult because it requires a more intimate knowledge of SAP and the business process being evaluated.

Future events — Do you need to be able to capture an event real-time and then delay its retrieval until a specified date? For example, an employee record may be updated today with a change of address that is effective three weeks from today. In this case, you may want to capture the event at the time of the update, but delay its retrieval until the effective date.

At this point, you should have an idea of the event detection mechanisms that you need to consider. Use Table 15 as a general guideline in determining which mechanism can be used for each business process you need to support.

Table 15. Event detection mechanism decision table

	Code enhancement	Batch program	Business workflow	Change pointer
Availability	High	High	Low	Low
Real-time integration	Yes	No	Yes	Yes
Reliability	High	High	Low	Medium
Flexibility	High	Medium	Low	Medium
Upgrade dependency	Low	Low	Medium	Medium
Difficulty	Medium	Low	Medium	High
Future Events	Yes	Yes	No	No

A final consideration to note is the development methodology of your site. Perhaps event detection using only business workflow is the preferred method and code enhancement cannot be used at all.

Using code enhancement is the recommended approach for event detection because it is reliable, highly flexible, synchronous, and has high availability. In contrast, business workflow and change pointer mechanisms are not generally available for all business processes. Batch program is typically used when real-time integration is not desired.

Each event detection mechanism has advantages and disadvantages for detecting an event in a business process. The following sections give more detail about each of the event detection mechanisms, including the main advantages and disadvantages of each.

All of the event detection mechanisms support real-time triggering and retrieval of events. However, only code enhancement and batch program provides the additional functionality of delayed retrieval. An event specified to be retrieved at a later date is called a future event.

Code enhancement

Code enhancement is implemented at specific points in the code of an SAP transaction. By making use of user exits, you can insert event detection code at the most logical point in a transaction. The event detection code allows for evaluation of criteria to determine whether an event is generated.

The general strategy of this mechanism is to insert your event detection code when the data for a transaction is about to be committed to the database.

Advantages

- Has access to SAP transactional information for the event detection process
- Allows the insertion of event detection code at an appropriate point of a transaction
- Provides synchronous event detection
- Limits the reliance on SAP functionality, so maintenance and enhancements are easier
- Supports future events

Disadvantages

- User exits may not always be in the appropriate location in the transaction
- SAP modification features may be necessary

Batch program

Batch program is useful when a large number of events of the same type (such as customer orders) need to be triggered, or a business process requires a large amount of processing time. This mechanism does not require any modifications to SAP-delivered code; however, you need to use (write) an ABAP program that evaluates criteria for detecting events.

Advantages

- Can be implemented for most business processes
- Accurately detects events
- Is easy to implement

- Can be scheduled to run at a specific time if runtime resources are an issue
- Supports future events

Disadvantages

- It does not provide synchronous event detection
- SAP transactional information is not available
- State (Create, Update, or Delete) or status changes cannot be detected or may not be easily detected
- If a background job is created to automate a batch program, an additional task needs to be maintained and monitored

Business workflow

Business workflow is a cross-application tool within the SAP application that enables you to integrate business tasks between applications. This tool supplements the existing business functions of the SAP application. The standard functions of SAP can be adapted using business workflow to meet the specific requirements of the desired business function. business workflow uses the Business Object Repository (BOR), which stores the definitions for each SAP object in the application.

Advantages

- Provides synchronous event detection
- Makes use of SAP's object-oriented business object capability to link the detection of events to ABAP function modules
- Is easy to implement

Disadvantages

- An SAP object does not exist in the SAP BOR for every business process
- The SAP event (such as Created or Deleted) may not exist for the SAP object
- It may not detect all changes in a business process
- It does not always provide the flexibility for detecting events at the proper time
- It depends on SAP-provided functionality, which may change between versions of SAP

Change pointer

Change pointer is a related feature of business workflow that uses change documents to detect events. Change documents are created for some business processes so that all changes for that business process are captured.

Advantages

- Provides synchronous event detection
- Needs only one SAP modification for an adapter function module to handle all business processes
- Generally available for the Logistics module
- Has access to SAP change pointer information for the event detection process
- If change documents are already used for a business process, it needs only a minimal amount of work to detect an event

Disadvantages

- It is somewhat flexible, but the event detection placement cannot be changed since it is done by SAP

- It requires a solid understanding of change documents and the business workflow environment
- You must do an SAP modification to turn on the change document flag for an SAP data element
- Change pointer information in SAP may not be sufficient for the event detection process

Implementing an event detection mechanism

After you determine the business process to support (for example, sales quotes or sales orders), and determine the preferred event detection mechanism, implement the mechanism for your business process.

Note: When implementing an event detection mechanism, it is a good idea to support all of the functionality for a business process in one mechanism. This limits the impact in the SAP application and makes event detection easier to manage.

The following sections describe the implementation process for the four event detection mechanisms implemented by the IBM WebSphere Business Integration Adapter for mySAP.com. Whenever applicable, an example is provided along with sample code.

Code enhancement

Code enhancement requires encapsulating a portion of ABAP code in a custom function module. The event detection code is written as a function module to ensure that the processing remains separate from the transaction. Any tables or variables used from the transaction need to be passed to the function module by value and not by reference.

To minimize the effects of locking a business object when retrieving an event, the function module typically executes in an update-task mode. To avoid inconsistencies, do not use update task if the function module is already being called within a process that is in an update -task mode.

To minimize the impact in the transaction, place the function module within another include program. Using an include program allows you to make changes to custom code rather than to SAP code.

The event detection code contains logic that identifies the object for the event. For example, the sales order transaction handles many types of orders, but only one order type is required. This logic is in the event detection code. The general strategy for placing this event detection code is to insert it just before the data is committed to the database. The function module containing the event detection code is typically created as a part of the function group for the business object.

To implement Code enhancement for event detection:

- Determine which verbs to support: Create, Update, or Delete. This helps define which transactions to investigate.
- Determine the business object key for the transaction. This key must be unique to allow the connector to retrieve the business object from the database. If a composite key is required, at triggering time you can specify each key attribute and its corresponding value as a name-value pair. When the business object is

created at polling time, the connector automatically populates the attributes with their values. For more information, see "Coding composite keys as name-value pairs" on page 95.

- Check that an SAP-provided user exit in the transaction has all of the information needed to detect an event. For example, a user exit may not be able to implement a Delete verb because the business object is removed from the database prior to that point.
- If a user exit cannot be used, determine the appropriate location for the event detection code, and then add the event detection code using an SAP modification. Select a location that has access to the business object key and other variables used to make the decision.

  If you are implementing the future events capability, in addition to adding the event detection code for future events, contact your BASIS administrator to schedule the adapter-delivered batch program `/CWLD/SUBMIT_IN_FUTURE` to run once every day.
- Research a business process by looking for a "`commit work statement`" in the code executed by the transaction for the business process. You can use the ABAP debugger to investigate the value of different attributes at that point.
- Determine the criteria for detecting an event.
- Create the function module containing the event detection code.
- Create the include program and then add it to the transaction's code. Test all of the scenarios designed to detect an event.

The following steps describe the process of creating an example SAP sales quote using the code enhancement event detection mechanism. The code that follows it is a result of this process.

1. Upon investigation of the SAP sales quote transaction, transaction VA21 is found to support the desired sales quote creation business process.
2. The sales quote number is determined to be the unique key. The Sales quote number is stored in table/field `VBAK-VBELN`.

   Note: Because this event uses a single unique key, the code example uses the `OBJKEY` parameter to pass the key's value. For an example of coding an event that uses a composite key, see "Coding composite keys as name-value pairs" on page 95.
3. Transaction VA21 has a user exit in the transaction flow as part of the document save process (`Form Userexit_save_document`). At this point in the transaction, the quote number is available when the user exit is executed.
4. The user exit belongs to other business processes, so additional coding is needed to differentiate a sales quote from other categories of documents. `VBAK-VBTYP` is available to determine the document category. A sales quote is saved in the SAP database with a document category of `B`.
5. An include statement is added to the user exit that points to the include program.
6. At this time, the include program and a function module need to be created.

/CWLD/ADD_TO_QUEUE: single key value example

The following code fragment illustrates the function call to the `/CWLD/ADD_TO_QUEUE` event trigger (using a single key value).

```
If VBAK-VBTYP = 'B'.
      C_OBJ_ORDER = 'SAP4_SalesQuote'.
         TMP_OBJKEY = XVBAK-VBELN.
    TMP_EVENT = 'Create'.
```

```
      CALL FUNCTION '/CWLD/ADD_TO_QUEUE'
            EXPORTING
                  OBJ_NAME               = C_OBJ_ORDER
                  OBJKEY                 = TMP_OBJKEY
                  EVENT                  = TMP_EVENT
                  GENERIC_RECTYPE = ''
            IMPORTING
                  RECTYPE                = TMP_RECTYPE
            TABLES
                  EVENT_CONTAINER = TMP_EVENT_CONTAINER
            EXCEPTIONS
                  OTHERS                 = 1.

Endif.
```

/CWLD/ADD_TO_QUEUE_IN_FUTURE: single key value example

The following code fragment illustrates the function call to the /CWLD/ADD_TO_QUEUE_IN_FUTURE event trigger (single key value).

```
DATA: DATE_IN_FUTURE LIKE SY_DATUM.

DATE_IN_FUTURE = VBAK-VDATU.

If VBAK-VBTYP = 'B'.
      C_OBJ_ORDER = 'SAP4_SalesQuote'.
         TMP_OBJKEY = XVBAK-VBELN.
    TMP_EVENT = 'Create'.

      CALL FUNCTION '/CWLD/ADD_TO_QUEUE_IN_FUTURE'
            EXPORTING
                  OBJ_NAME               = C_OBJ_ORDER
                  OBJKEY                 = TMP_OBJKEY
                  EVENT                  = TMP_EVENT
                  VALID_DATE             = DATE_IN_FUTURE
            IMPORTING
                  RECTYPE                = TMP_RECTYPE
            TABLES
                  EVENT_CONTAINER        = TMP_EVENT_CONTAINER
            EXCEPTIONS
                  OTHERS                 = 1.

Endif.
```

Coding composite keys as name-value pairs

If an event's key is composed of multiple fields rather than a single key field, you can specify the name of each key attribute and its corresponding value. Because you specify the attribute's name, the attribute need not be marked as IsKey for the connector to populate it and use it for retrieval.

If you specify more than one name-value pair, the connector sets the value of multiple attributes in the business object it creates to retrieve the full object from the application. If you specify a single name-value pair, the connector sets the value of the specified attribute rather than the first attribute that is marked IsKey.

Because IDoc handlers do not use name-value pairs, it is important that you not specify name-value pairs when using /CWLD/IDOC_HANDLER. For more information, see "IDoc handlers and the retrieve verbs" on page 85.

The following steps describe the process of creating an example SAP sales quote that uses three fields in its composite key. The code that follows it is a result of this process.

1. Create a local name_value_pairs internal table based on the structure (/CWLD/NAME_VALUE_PAIRS) delivered with the adapter. This structure has two columns: ATTR_NAME and ATTR_VALUE.
2. Before calling the function module /CWLD/ADD_TO_QUEUE or /CWLD/ADD_TO_QUEUE_IN_FUTURE, write code that adds the names of the key attributes and their values to your internal table.
3. Change the function module /CWLD/ADD_TO_QUEUE or /CWLD/ADD_TO_QUEUE_IN_FUTURE:
   - Because you will not be using the OBJKEY parameter to pass the key's value, comment out the line for this parameter.
   - Because you will be using the NAME_VALUE_PAIRS table to pass the composite key's value, add a line for this table.
4. The triggering function automatically formats each event key. The format uses the following syntax:

   attribute1=value1|Cx|*attribute2=value2*|Cx|[*attributeN=valueN*|Cx|]

   where:

attribute	The name of the key attribute (not case-sensitive)
value	The value of the key attribute (case-sensitive)
|Cx|	Terminator for each name-value pair (used even if only one name-value pair is specified)

The order in which you specify name-value pairs in your code need not match the order of the attributes in the business object. However, the event fails if you specify an attribute that does not exist in the business object.

The following code fragment specifies, at the time of triggering, the customer number, sales organization, and distribution channel in table KNVV as name-value pairs. Two lines are highlighted in the code for the function module /CWLD/ADD_TO_QUEUE:

- The line that passes a value to the OBJKEY parameter (commented out)
- The line that specifies the NAME_VALUE_PAIRS table

```
DATA:  name_value_pairs LIKE /cwld/name_value_pairs OCCURS 5 with header line.

MOVE   'CustomerId' TO name_value_pairs-attr_name.
MOVE   knvv-kunnr TO name_value_pairs-attr_value.
APPEND name_value_pairs.

MOVE   'SalesOrg' TO name_value_pairs-attr_name.
MOVE   knvv-vkorg TO name_value_pairs-attr_value.
APPEND name_value_pairs.

MOVE   'DistributionChannel' TO name_value_pairs-attr_name.
MOVE   knvv-vtweg TO name_value_pairs-attr_value.
APPEND name_value_pairs.
If VBAK-VBTYP = 'B'.
      C_OBJ_ORDER = 'SAP4_SalesQuote'.
         TMP_OBJKEY = XVBAK-VBELN.
    TMP_EVENT = 'Create'.

      CALL FUNCTION '/CWLD/ADD_TO_QUEUE'
           EXPORTING
                OBJ_NAME               = C_OBJ_ORDER
*               OBJKEY                 = TMP_OBJKEY
                EVENT                  = TMP_EVENT
                GENERIC_RECTYPE = ''
           IMPORTING
```

```
      RECTYPE                 = TMP_RECTYPE
TABLES
      NAME_VALUE_PAIRS = name_value_pairs
      EVENT_CONTAINER = TMP_EVENT_CONTAINER
EXCEPTIONS
      OTHERS                  = 1.

Endif.
```

Batch program

To implement batch program as an event detection mechanism, you must write an ABAP program that evaluates database information. If the criteria in the ABAP program is fulfilled when the program executes, then an event is triggered.

To implement batch program for event detection:

- Determine which verb to support: Create, Update, or Delete.
- Determine the business object key for the transaction. The business object key must be unique so that the business object can be retrieved from the database. A composite key may be required. For example, implementing a batch program for inventory levels of materials at different plants requires the key `Material_key + Plant_key`.
- Determine the criteria for detecting an event. You should have knowledge of the database tables associated with a business object.
- Create an ABAP program containing the criteria for generating an event.
- If you are implementing the future events capability, in addition to adding the event detection code for future events, contact your BASIS administrator to schedule the adapter-delivered batch program `/CWLD/SUBMIT_IN_FUTURE` to run once every day.

  See "/CWLD/ADD_TO_QUEUE_IN_FUTURE: single key value example" on page 95 for example code that implements the future events capability.
- Determine if a background job is required to automate the batch program. A background job is useful if there is an impact on system resources, which makes it necessary to run the batch program during off-peak hours.

The following steps describe the process of creating a batch program that detects events for all sales quotes created on today's date. The code that follows it is a result of this process.

1. Create is determined to be the supported verb.
2. The quote number is determined to be the unique key used to retrieve the events.
3. The creation date (`VBAK-ERDAT`) and the document category (`VBAK-VBTYP`) need to be checked.

The following sample code supports the SAP sales quote as a batch program:

```
REPORT ZSALESORDERBATCH.

tables: vbak.

parameter: d_date like sy-datum default sy-datum.

data: tmp_key like /CWLD/LOG_HEADER-OBJ_KEY,
      tmp_event_container like swcont occurs 0.

" retrieve all sales quotes for today's date
" sales quotes have vbtyp = B
select * from vbak where erdat = d_date
```

```
                    and vbtyp = 'B'.

  tmp_key = vbak-vbeln.

  CALL FUNCTION '/CWLD/ADD_TO_QUEUE'
       EXPORTING
            OBJ_NAME        = 'SAP4_SalesQuote'
            OBJKEY          =  tmp_key
            EVENT           = 'Create'
            GENERIC_RECTYPE = ''
       IMPORTING
            RECTYPE         = r_rectype
       TABLES
            EVENT_CONTAINER = tmp_event_container.

  write: / vbak-vbeln.
endselect.
```

Business workflow

Business workflow is a set or sequence of logically related business operations. The processing logic within a workflow detects events. The business workflow event detection mechanism relies on the SAP Business Object Repository (BOR), which contains the directory of objects along with their related attributes, methods, and events.

To implement business workflow for event detection:

- Determine which SAP business object represents the functionality that you need. Check if the events trigger, start, or end a workflow. You can use the Business Object Builder (transaction SWO1) to search for the appropriate business object.
- Create a subtype of this SAP business object. A subtype inherits the properties of the supertype and can be customized for use.
- Activate the events (such as CREATED, CHANGED, and DELETED) for the business object by customizing the subtype.

The following example of SAP sales quote can be used to implement an event trigger using business workflow:

1. Search the BOR for the appropriate sales quote business object. A search can be done using the short description field and the string '*quot*'. BUS2031 (Customer Quotes) is one of the business objects returned.
2. Upon further investigation of BUS2031, it is determined that the key field is CustomerQuotation.SalesDocument (`VBAK-VBELN`).
3. A subtype for BUS2031 is created using the following entries:

   Object type—`ZMYQUOTE`

   Event—`SAP4_SalesQuote`

   Name—`SAP4 Sales Quote`

   Description—Example of an SAP 4 Sales Quote Subtype

   Program—`ZMYSALESQUOTE`

   Application—`V`
4. The event detection mechanism is activated by adding an entry to the Event Linkage table (transaction SWE3). The create event is activated using the following entries:

   Object type—`ZMYQUOTE`

   Event—`SAP4_SalesQuote`

   Receiver FM—`/CWLD/ADD_TO_QUEUE_DUMMY`

Receiver type FM—/CWLD/ADD_TO_QUEUE_WF

Note: The Receiver and Receiver type function modules (FM) point to /CWLD/ADD_TO_QUEUE. The DUMMY function module is used only because sometimes the SAP application requires that both fields be populated. The WF function module translates the SAP standard interface to the one used by /CWLD/ADD_TO_QUEUE.

The business workflow event detection mechanism is created and active. It is set up to detect all SAP Customer Quotes that are created.

Change pointer

Change pointer uses change documents and is one of the more challenging event detection mechanisms to implement. SAP's Business Object Repository (BOR) is used as well as Application Link Enabled (ALE) technology. A change document always refers to a business document object having at least one database table assigned to it. If the data element in a table is marked as requiring a change document and the table is assigned to a business document object, then a change in value of the field defined by the data element generates a change document. The changes are captured in tables CDHDR and CDPOS and are used for event detection.

To implement change pointer for event detection:

- Activate the global Change pointers flag in transaction BD61.
- Change the SAP function module CHANGE_POINTERS_CREATE to include the function module call to /CWLD/EVENT_FROM_CHANGE_POINTR.
- Determine which verbs to support: Create, Update, or Delete.
- Check if the SAP business process (transaction) utilizes change documents:
  - In the Environment menu for the transaction, does a Change function exist? How about when you click Go To, and then click Statistics?
  - If you change data in the transaction, is there a new entry in table CDHDR that reflects the change?
  - In the database tables associated with a transaction, do any of the data elements have the Change Document flag set?

  If the answer is Yes to any of these questions, the transaction uses change documents.
- Determine if the data elements that set the Change Document flag capture all of the information needed to detect an event. Changing the Change Document flag is not recommended because it changes an SAP-delivered object.
- Determine the business object key for the transaction. The business object key must be unique so that the business object can be retrieved from the database. A composite key may be required. This is normally table/field CDHDR-OBJECTID.
- Determine the criteria for detecting an event. Use table/field CDHDR-OBJECTCLAS as the main differentiator. CDPOS-TABNAME may also be used to detect the event.
- Update function module /CWLD/EVENT_FROM_CHANGE_POINTR with the logic to detect the event.

The following example of an SAP sales quote can be used to implement an event trigger using change pointer:

1. Update is determined to be the supported verb. Investigating the sales quote create transaction shows that the Create verb is not detected through this mechanism.

2. When performing the checks of the business for sales quote:
   - The Change function is available from the Environment menu in transaction VA22.
   - Making a change to a sales quote results in a new entry in table CDHDR.
   - Looking at table VBAP, the field ZMENG has the Change Document flag set.
3. No evaluation of data elements was done for this example.
4. The sales quote number is determined to be the unique key in CDHDR-OBJECTID.
5. CDHDR-OBJECTCLAS has a value of VERKBELEG, which is the main differentiator. Only sales quotes should be picked up. The code checks the TCODE field in the header table, but a proper lookup should be done in the VBAK table.

The following sample code is added to /CWLD/EVENT_FROM_CHANGE_POINTR:

```
when 'VERKBELEG'.
   data: skey    like /cwld/log_header-obj_key,
      s_event like swetypecou-event,
      r_genrectype like swetypecou-rectype,
      r_rectype like swetypecou-rectype,
      t_event_container like  swcont occurs 1 with header line.

   " Quick check. Should check document category (VBTYP) in VBAK.
   check header-tcode = 'VA22'.

   " Event detection has started
   perform log_create using c_log_normal c_blank
                            c_event_from_change_pointer c_blank.

   " Set the primary key
   skey = header-objectid.

   " Set the verb
   s_event = c_update_event.

   " Log adding the event to the queue
   perform log_update using c_information_log text-i44
                           'SAP4_SalesQuote' s_event skey.

      " Event detection has finished.
      perform log_update using c_finished_log c_blank
                               c_blank c_blank c_blank.

   call function '/CWLD/ADD_TO_QUEUE'
      exporting
            obj_name                 = 'SAP4_SalesQuote'
            objkey                   = skey
            event                    = s_event
            generic_rectype          = r_genrectype
      importing
            rectype                  = r_rectype
      tables
            event_container          = t_event_container
      exceptions
            others                    = 1.
```

Chapter 8. Testing a business object for the ABAP Extension module

Once you have developed an application-specific business object and a supporting ABAP handler, you must unit test to make sure they support the desired functionality. IBM provides unit test tools that operate independently from your WebSphere business integration system. This means that you do not need to have an integration broker or the connector running to test your business object. However, these tools do not replace full end-to-end testing through the WebSphere business integration system and are meant only to be used for unit testing of individual business objects and ABAP handlers.

This chapter contains the following sections:

Preparing to test

All business object processing originates from the Java component of the connector. This applies to all business objects and all possible verbs. To unit test, IBM provides an ABAP program that simulates the connector's action of sending in a business object request.

Specifically, the program simulates the `doVerbFor()` processing in the Java component of the connector by calling the ABAP function module `/CWLD/RFC_DO_VERB_NEXTGEN`. Like `doVerbFor()`, the test program requires a business object as an input to pass to the ABAP function module. The ABAP test program uses a text file as its input.

All input test files have the same ASCII text format. From this file format, the test program restructures the data to resemble the business object passed to `/CWLD/RFC_DO_VERB_NEXTGEN`. The following rules apply to business object input files:

- Business object must have only one parent business object in a file.
- Child business objects are ordered first in depth, then in breadth.
- Attributes and business objects must be ordered in the exact sequence that they occur in the business object repository definition.
- For each attribute, the information described in Table 16 must be provided in the described format and in the sequence shown (leading spaces after the "=" are ignored):

Table 16. Attribute properties and values

Attribute Property	Description or Possible Values
`Name`	name of the attribute
`Value`	value of the attribute or `CxIgnore = 'CxIgnore'`or `CxBlank = ' '`

Table 16. Attribute properties and values (continued)

Attribute Property	Description or Possible Values
`IsKey`	value that specifies whether the attribute is a key: `0` = no `1` = yes
`Peers`	*`NumberOfPeers`* value expressed an integer that represents the total number of child business objects at the same level For example, if an Item business object contains two line items, each line item would have the value '2'.
`AppInfo`	application-specific information that is specific to each business object

In addition to the test program, IBM also provides a program for generating the object test input file. The Test File Generator builds the test file based on one of several different inputs. Table 17 lists the Test File Generator options.

Table 17. Test file generator inputs and outputs

Option	Required inputs	Description of output
Dynamic Ret/Tran	Table entries in: • `/CWLD/WIZ_OUT` (Dynamic Retrieve) • `/CWLD/WIZ_IN` (Dynamic Transaction)	A test input file containing either a complete set or subset of attributes based on their net use in both Dynamic Retrieve and Dynamic Transaction tables. This is Dynamic Retrieve and Dynamic Transaction metadata.
IDoc Structure	An IDoc type defined in your SAP application	A test input file containing one instance of every possible object (segment) and attribute (segment-field). This is an IDoc structure.
IDoc object	• An ABAP function module developed to retrieve IBM WebSphere business objects • The object's key	A test input file with only the attributes (fields) and objects (segments) that exist for the particular object identified with the key. This is an IDoc object.
Repository Definition	The same text file that used to load the repository (when WebSphere InterChange Server is the integration broker) or copied to the repository (when WebSphere WMQ Integrator Broker is the integration broker)	A test input file containing one instance of every possible object and attribute. This is a repository definition.

Choosing to generate an IDoc object is useful because the Generate test file tool generates an input file that has valid attribute values, making it easier to test Create, Update, or Delete verbs because the values have been validated.

The other three options (Dynamic Ret/Tran, IDoc Structure, and CW Repository Definition) are similar because they provide input files with no attribute values.

Unit test issues

The unit test tools test all SAP development work that handles business object processing for the connector. Also, the unit test tools enable you to test the interaction of your work with the ABAP components of the connector. The test tools allow you to test your development work as an online user (real-time) only.

It is important to understand the differences between testing the connector as if operating as a background user and testing the connector as an online user. The main differences are as follows:

Memory

When testing a business object, the connector must log into the SAP application. In addition, logging in is required to test the business objects by generating events and using the test tools in IBM CrossWorlds Station.

The connector runs as a background user, so it processes in a single memory space that is never implicitly refreshed until the connector is stopped and then restarted (therefore it is critical in business object development to clear memory after processing is complete). Since you are an online user, memory is typically refreshed after each transaction you execute.

For more information, see Chapter 6, "Developing business objects for the ABAP Extension module," on page 69. Any problems that may occur because of this (for example, return codes never being initialized) are not detected using the test tool; only testing with the connector reveals these issues.

Screen flow behavior

Screen flow behavior is relevant only when using the Call Transaction API. The precise screen and sequence of screens that a user interacts with is usually determined at runtime by the transaction's code. For example, if a user chooses to extend a material master record to include a sales view by checking the Sales view check box, SAP queries the user for the specific Sales Organization information by presenting an additional input field. So, the transaction source code at runtime determines the specific screen and its requirements based on the data input by the user. While the test tool can handle this type of test scenario, there is a related scenario that the test tool cannot handle.

SAP's transaction code may present different screens to background users and online users (usually for usability rather than performance). The test tool operates only as an online user. The connector operates only as a background user. Despite this difference, unit testing allows you to handle most testing situations.

Testing an ABAP handler

To test an ABAP handler, you must first generate a business object input file. You may need to modify the file to contain attribute values and the appropriate application-specific information. Once you are ready, all you need to do is execute the test program by pointing to your test file as input.

Creating a test file

To create a test file:

1. Go to IBM CrossWorlds Station (transaction `/n/CWLD/HOME`), and then click the Tools tab.
2. Click the Create Test File button under Test Tools.
3. Enter the name of your business object and the verb you plan to test.
4. Select the source definition on which to base your input file. See Table 17 for a description of the test file options.
5. Enter the additional data required for the source.
6. Click the Generate button.

   A dialog box appear asking if you want to modify the test file. This is your opportunity to edit the test file in SAP's editor. If you do not wish to edit at this time, click no; otherwise, click Yes, and then click the Back arrow (F3)when finished.
7. Enter a filename and location to save your test file. It is recommended that you use the naming convention `Object_verb.in`.
8. Once you have saved your generated test file, you must open the file in a test editor and do the following:
   - Modify the verb application-specific information to point to your ABAP handler. For example, `:function1:function2`.

     For more information on the proper syntax, see "Business object data routing to ABAP handlers" on page 61.
   - Verify that the appropriate attribute on the parent business object is marked `isKey`.
   - Add input values for the attributes, as required.

Using the test file

Once you have created the test file, you are now ready to use it to test the business object. To use the test file:

1. Go to IBM CrossWorlds Station (transaction `/n/CWLD/HOME`), and then click the Tools tab.
2. Click the Test Program button under Test Tools.
3. In the input file field, enter the location and filename of your input file.
4. If you want to generate output data, enter a filename and location for the output data. The filename can be the same as the input filename, however, it will overwrite the input data. This step is optional.
5. Click the Execute button.

When finished, the program displays the last message that was raised during processing. In addition, the processed data is displayed on the screen for verification. This is the same information that is generated in the output file of step 4.

You can look in the adapter's ABAP Log for additional details.

Chapter 9. Managing the ABAP Extension module

The IBM CrossWorlds Station tool (transaction /n/CWLD/HOME) enables you to maintain the IBM WebSphere Business Integration Adapter for mySAP.com (SAP R/3 Version 4.x) for event processing. You can also use this tool to maintain the connection to the SAP application. You can view the connector log file and the SAP Gateway Service connections. Also, you can reprocess archived objects from the connector log, view events waiting to be processed, schedule specific events to be processed at a later time, and resubmit and delete events from the archive table.

This chapter contains the following sections:

Managing the connector log file

The connector log in the SAP application displays in reverse chronological order all events and errors that relate to the connector, such as Create or Update operations, or events that arrive in the event queue. The log file lists the date, time, and event for each log entry. The log file is a good source to start troubleshooting problems.

Setting log options

You can set the global and user settings to the level of detail you want logged in the connector log file, as well as the number of entries and type of data you want displayed. To set the connector logging levels using IBM CrossWorlds Station, click the Configuration tab, and then select from level 0 - 3 under Logging Level.

The four levels of logging are as follows:

- 0 — Off
- 1 — Log only warnings and errors
- 2 — Log every event with minimal information
- 3 — Log each event in detail, including every attribute of every business object

Note: Logging level 0 is not recommended. Logging level 1 is recommended for a production system. Logging level 3 is recommended for a development or debugging system.

Displaying the log

To view recently processed objects and details associated with them, display the connector log. To display the connector log in the SAP application:

1. Go to IBM CrossWorlds Station (transaction /n/CWLD/HOME).
2. Click the Management tab, and then click the Log button.

Log entries display the date, time, and event. Entries are color-coded:

green—indicates a successful event

yellow—indicates a warning message

red—indicates an error

white— indicates an archived object

Magenta (SAP application GUI versions previous to 4.6) or orange (SAP application GUI version 4.6 and later) entries provide information on the beginning and end of the event. Click on any arrow to link to SAP's display transaction for that business object.

Filtering log details

You can change the amount of detail that is displayed about each event. To change the display level, click the More Details or Fewer Details button depending on the level of detail desired.

If the amount of data displayed is more than you currently need, narrow the information displayed. For example, you can view business objects by user, name, date, or log entry number.

1. Click the Filter Data button.
2. Populate the appropriate fields to filter the log file.
3. Click Filter.

In the Configuration tab, you can set user settings for the number of log entries to display at one time and the default logging display level.

Reprocessing archived objects

You can reprocess failed or archived objects from the connector log file. Failed objects are objects in SAP that fail to process successfully. Archived objects are objects that you configure to be archived without processing. In either case, you can manually step through the object by setting breakpoints in specific locations of the code. For Dynamic Transaction and IDoc objects only, you can step through the screens for the transaction.

The breakpoints can be set before the:

- Function module `/CWLD/RFC_DO_VERB_NEXTGEN` is called
- First function module executes
- Main processing step executes

  The placement of the breakpoint is different depending on the type of object.
  - Dynamic Retrieve—Before the Select statement
  - Dynamic Transaction—Before the Call Transaction statement
  - IDoc—Before the IDoc function module is called
  - BAPI— Before the BAPI-Wrapper function module is called

Dynamic Transaction and IDoc objects use call transactions; therefore you can view the screen processing for these objects. You have the option of viewing:

- All screens
- Only the screens with errors
- None of the screens

Dynamic Retrieve and BAPI objects do not use screen processing.

Configuring an object to be archived

By default, ABAP Extension Module business objects that have no archive option (`A`, `X`, or `N`) specified in their verb's application-specific information are archived in case of failures. In other words, when processing yields return codes other than 0 or 21, the business objects are archived in the `/cwld/obj_arc_h` and `/cwld/obj_arc_i` tables.

Important: Because these archive tables grow, they must have their contents deleted or archived periodically to prevent impacting overall database performance.

Altering the archiving behavior is accomplished at the business object's verb level; that is, for each business object, the archiving activity can vary by verb. To specify how an object is archived, use the following syntax in the verb's application-specific information:

```
AppSpecificInfo = connectormodule.class, ArchiveParameter: ABAPhandler
```

where `ArchiveParameter`:

`A` Archives the object when it first enters the SAP application.

`N` Suppresses object archiving. Even in the case of failure, the object is not archived.

`X` Archives the object immediately. The log is updated with a warning message stating that processing ended. A success code is returned to the connector, so that the requesting integration broker processes successfully.

You can specify more than one parameter at a time. The `A` and `X` archive parameters add an entry in the log table with a link to the reprocessing tool in IBM CrossWorlds Station. The status of the archived object is entered in the line below the entry for the archived business object.

The following example archives a Dynamic Transaction object and adds a entry in the log table.

```
AppSpecificInfo = sap.sapextensionmodule.VSapBOHandler,
A:/CWLD/DYNAMIC_RETRIEVE
```

The following example archives an IDoc object, SAP4_Order Create, when it enters the SAP application, and then stops the processing of the object.

```
AppSpecificInfo = sap.sapextensionmodule.VSapBOHandler,
X:/CWLD/ORDER:ORDER_C1
```

Note: In your production environments, use only the `N` parameter for business objects and all their verbs. When WebSphere InterChange Server is the integration broker, you should only use System Manager to reprocess and resubmit business objects; you should *not* use the IBM CrossWorlds Station reprocessing tools in your SAP application.

Using the reprocessing tool

The Reprocessing Tool enables you to reprocess WebSphere business objects for SAP using the ABAP Debugger.

Attention: This tool should be used in a development environment only.

- During development and testing, you can specify that certain business objects are archived as they arrive into the SAP application, and then process these business objects using the ABAP Debugger.

- You can reprocess the same business object as many times as you want. A business object is always available for reprocessing until it is deleted.

To reprocess archived objects:

1. Go to the connector's log in the SAP R/3 application.
2. Double-click the archived object entry.

   The "CW reprocess objects from archive tables" window appears. Its Archived Object Number field is populated with the object number.
3. Click the Set Breakpoint check boxes for the breakpoints that you want to set. You can set multiple breakpoints if needed.
4. For objects that use Call Transaction, you can select the screen processing option.
5. Click Execute (F8).

   The ABAP Debugger is invoked with the archived object.
6. Use the ABAP Debugger to step through the object.

To manually access the Reprocessing Tool in IBM CrossWorlds Station, from the Tools tab click Reprocess Object. Enter the appropriate values in the fields provided.

Deleting archived objects

You can delete archived objects from the SAP R/3 application using the adapter-provided Delete Archive Objects tool. This tool enables you to delete archived objects manually. Once you have deleted an archived object, the object's entry in the connector log is updated with the new status. The object is physically deleted and only the status of the object is kept for reference.

To delete an archived object using IBM CrossWorlds Station (transaction `/n/CWLD/HOME`):

1. From the Maintenance tab, click the Del Object Archive button.
2. Specify the objects to be deleted. You can delete objects based on the following:
   - Archive number
   - Object name
   - User (connector name)
   - Creation date
   - Status
3. Click Execute (F8).

To schedule an archive object program to delete objects automatically, contact your basis administrator and schedule report `/CWLD/DELETE_OBJECT_ARCHIVE`.

Setting up truncation of the event log

SAP keeps an event log of the connector's activity. This log can, over time, take up a lot of disk space. To save disk space, you can set this log to automatically truncate. When you set automatic truncation, by default SAP prints the truncated entries to the default printer of the user who sets up the job. Therefore, you may also want to control the print options.

To truncate the log manually:

1. Go to IBM CrossWorlds Station (transaction `/n/CWLD/HOME`).
2. Click the Maintenance tab.

3. In the Online section, click Delete Log.
4. Populate the applicable fields.
5. Click the Execute button (F8).

To schedule the automatic truncation of the event log, set up the truncation options, and contact your basis administrator to schedule report /CWLD/DELETE_LOG.

Important: It is recommended that you run this report on a regular basis.

Monitoring the SAP gateway service connections

You can monitor the SAP gateway service connections between the connector and the SAP application. Each entry displays information such as connector host name, user name, and connection status.

To monitor the SAP Gateway Service connections:

1. Go to IBM CrossWorlds Station (transaction /n/CWLD/HOME).
2. Click the Management tab, and then click Gateway.
3. Click on a server name to view more details.

Shutting down the connector

It is recommended that you shut down your connector using System Manager (when WebSphere InterChange Server is the integration broker).

Important: Do not use the Gateway monitor window. If you use the Gateway monitor window, your connector may not shut down properly.

Maintaining the event queue

You can check the outgoing current event queue for events that have not been processed by the connector.

1. Go to IBM CrossWorlds Station (transaction /n/CWLD/HOME).
2. Click the Management tab, and then click Current Events.
3. Click the Execute button (F8) to display the current event queue.

To limit the number of event entries that are displayed, populate the applicable fields in the Current Event Selection section. For example, to limit the displayed entries for a particular business object, enter a business object name in the Object Name field. If you do not know the exact syntax for the business object name, click the Object Name field, click the arrow button (F4), and then select the appropriate business object name.

To see more information about an event, double-click an event field. Under normal conditions, events are picked up every few seconds. If an event is displayed, it has not been processed by the connector. This may indicate that the connector is not running.

The following is a list of the possible event status values for the event queue:

P — Prequeued	When an event is triggered, the status is initially set to prequeue (P), since it has not yet been determined whether the business object is locked.

L — Locked	When a user creates or updates a business object in SAP, a lock is placed against that business object. Once the business object has been committed to the database, SAP removes the lock. If an event is triggered while a business object is locked, the event remains in the event queue with status locked (L) until the lock has been removed.
Q — Queued	When the business object is no longer locked, the status changes to queued (Q), and the event is ready to be picked up by the connector. The event remains in this status until a confirmation of a retrieval is received.
R — Retrieved	When the business object is retrieved it is marked with an R in the event queue. The event remains in the queue until the event has been processed.

Maintaining the archive table

Using the IBM CrossWorlds Station tool, you can display the archive table and determine the status of archived events. In the table, you can identify events that need to be resubmitted for polling when an integration broker subscribes to them.

To display the archive table:

1. Go to IBM CrossWorlds Station (transaction `/n/CWLD/HOME`).
2. Click the Management tab, and then click Archived Events.
3. Click the Execute button (F8) to display the archive queue.

To limit the number of archive entries that are displayed, populate the applicable fields in the Archived Event Selection section. For example, to limit the displayed entries for a particular business object, enter a business object name in the Object Name field. If you do not know the exact syntax for the business object name, click the Object Name field, click the arrow button, and then select the appropriate business object name.

To see more information about an event, double-click an event field. The following is a list of the possible event status values for the archive table:

0 — Success	The connector successfully processed the event and sent the business object to the integration broker.
1 — Error in SAP	The connector encountered an error while retrieving the business object within SAP for this event.
2— Not Subscribed	No integration broker was subscribed to the combination of the business object and verb for this event.
3— Error in Java	The connector encountered an error during one of the following: • Receiving the business object from SAP • Converting the SAP business object to a WebSphere business object for SAP • Inserting the business object into the message queue
4 — Max requeued	The event was requeued more than the maximum times specified by the requeued constant, `c_maximum_requeue` (usually 100). An event is requeued if its business object is locked.
5 — Multiple Event	Some business objects have single events in the event table that cause multiple events to be created at the time of retrieval. The original single event does not create a business object and is therefore archived using this event status.
6— Event Deleted	A user manually deleted the event from the event table.

Resubmitting events from the archive table

You can resubmit events from the archive table to the event queue for reprocessing. Depending on how you want to handle the events in the archive table, you have the option of resubmitting a single event or multiple events. Keep in mind that resubmitting events only moves the events from the archive table to the event table and therefore the events do not pass through event distribution, event restriction, or event priority. Follow these steps from the Archived Events window:

1. Click the Execute button (F8) to display the archive queue.
2. Select the events to be resubmitted.
3. Click the Resubmit button, or from the Archive Entry menu, click Resubmit (F8).

A status message displays. You can display the connector log to view the event and its new status.

Deleting events from the archive table

You can delete archive events manually or schedule them to be deleted automatically.

To delete archive events manually:

1. Go to IBM CrossWorlds Station (transaction `/n/CWLD/HOME`)
2. Click the Maintenance tab.
3. In the Online section, click Delete Event Archive.
4. Populate the applicable fields.
5. Click the Execute button (F8).

To schedule automatic deletion of archive events, contact your basis administrator and schedule report `/CWLD/TRUN_EVENT_ARCHIVE_TAB`.

Chapter 10. Upgrading the ABAP Extension module

This chapter describes the upgrade process for the ABAP Extension Module. It assumes that you are not modifying the repository definitions for the connector or any objects unless explicitly stated to do so. This chapter focuses on the ABAP components of ABAP Extension Module of the IBM WebSphere Business Integration Adapter for mySAP.com.

When upgrading, you must have the latest ABAP Extension Module components for your version of SAP R/3 version. The goal of the upgrade process is to get your ABAP handler development to work with the latest ABAP Extension Module components.

Upgrading the ABAP Extension Module can be described in two distinct scenarios:

- Upgrading an SAP R/3 system that contains adapter-provided ABAP handlers

  For example, you may be running an SAP R/3 version 3.1 system that you want to upgrade to SAP R/3 version 4.6. After you upgrade the SAP R/3 system, you must upgrade the adapter environment. For details on upgrading the adapter environment in a new version of SAP R/3, see "Upgrading within a new version of SAP R/3" on page 115.
- Implementing an adapter-provided ABAP handler for an object that supports an older version of SAP R/3

  For example, you may be using the connector that supports the SAP R/3 version 4.6 application and want to use the Material object that supports SAP R/3 version 4.0 or 4.5. To use this Material object, you need to upgrade it to your SAP R/3 version 4.6 system. For details on how to upgrade an object to a newer version of SAP R/3, see "Upgrading ABAP handlers" on page 116.

Upgrading within a new version of SAP R/3

The upgrade process for the SAP R/3 application does not modify any of the adapter's ABAP development, but it may modify the SAP R/3 application so that some of the adapter's ABAP development does not work properly.

This section describes how to upgrade the adapter's ABAP development in an upgraded SAP R/3 application. Before you can upgrade the adapter, you must have already upgraded your SAP R/3 application.

To upgrade the adapter's ABAP development:

1. Install the latest ABAP Extension Module transport files for the correct version of the SAP R/3 application.

   You must install the correct version-specific transport files. For details on installing these transport files, see "Connector transport file installation" on page 47.
2. Compile all programs and resolve syntax errors associated with the ABAP development.

The easiest way to find syntax errors is to generate each function group associated with each object and fix the errors one at time. Repeat this process until all function groups compile successfully. Be sure to generate any other programs such as triggering programs that are not associated with a function group.

If you are upgrading to SAP R/3 version 4.x, note that the 4.x ABAP handlers use the product namespace /CWLD/. For special considerations for upgrading to the connector supporting SAP R/3 version 4.x, see "Connector for SAP R/3" on page 118.

3. Test the new environment and make modifications as needed.

   Only a full system test enables you to work out any problems with the upgrade. Test your event detection mechanisms by running the appropriate transaction or program and sending business objects to the SAP system. Use the adapter's log within the SAP system to help identify other issues.

   For information on testing business objects, see Chapter 8, "Testing a business object for the ABAP Extension module," on page 101.

Upgrading ABAP handlers

Upgrading ABAP handlers has two steps.

1. Resolve any compilation errors that may arise when introducing your ABAP handlers into an environment with a different version of the ABAP Extension Module.
2. Evaluate the functionality that the business object provides in the newer SAP R/3 version. For example, the business object may operate properly but may not return the right information; or maybe the business object no longer functions because SAP has changed the screens for the Call Transaction.

This section details the processes of the first step, such as packaging the business object's ABAP handler and providing guidelines for possible compilation conflict points. The second step is not addressed in this section. See Chapter 6, "Developing business objects for the ABAP Extension module," on page 69 for more information on extending the functionality of your objects.

Attention: Once you upgrade an object, it is considered custom work even if it was originally developed by IBM.

Upgrade ABAP handlers when:

- You want to use a previously implemented IBM-delivered SAP R/3 business object in a later version of SAP R/3. For example, you may have already implemented a Customer business object in your SAP R/3 version 3.x system that does not exist in the 4.6 system.
- You want to use an adapter-provided SAP R/3 object that supports an SAP R/3 version other than the version that you need. For example, you may want to use the adapter-provided Material business object for SAP R/3 version 3.x in your SAP R/3 version 4.6 system.

Essentially the upgrade procedure is the same. The only difference is that upgrading a previously implemented business object requires you to package the business object into a transport file as the second step.

Note: If you have objects in SAP R/3 version 4.6 that do not take advantage of the IBM product namespace, you need to upgrade those objects to the namespace.

To upgrade an adapter-provide ABAP handler from one SAP R/3 version to another:

1. Verify that the latest version of the ABAP Extension Module transport files for your version of SAP R/3 are installed.
2. Package existing business objects into transport files. Note that if you are upgrading an business object that has not been modified for your implementation, skip to step 3, because you should be able to use the original transport that was loaded.

   Use the adapter-delivered transport files as templates for what should be included for each business object. This may include function groups, IDoc definitions, and Dynamic Retrieve and Dynamic Transaction data.

   - Include any additional programs and custom work.

     Custom work done in the ABAP component of the connector needs to be manually applied to the new SAP R/3 ABAP component of the connector. For example, you need to manually apply any changes to adapter-provided ABAP handlers such as IDoc Handler or Dynamic Transaction.
   - Check to see if changes were made to program `YXRRESTR`. This program is intended for customer modification.

     If changes were made, you can avoid conflicts by downloading the custom work as text files, not as transport files. Use the corresponding old program as a reference for updating `/CWLD/TRIGGERING_RESTRICTIONS`.
   - Release the transports and note the transport numbers. The BASIS administrator needs this information to load the objects in the new SAP R/3 system.
3. For IDocs (that define ABAP handler business objects) in an SAP R/3 version 3.x system only, capture the structure and segment definitions of the IDocs and then manually re-create them in the new system.

   If you do not have an SAP R/3 version 3.x environment and IDocs, then skip this step.
4. Install the business object transport files. You should have your local BASIS administrator install the transports for the business objects you packaged in step 1.

   The BASIS administrator should use all of the override codes available for the transport. This forces the business objects into the environment even if there are compilation errors. Before importing the business objects, the BASIS administrator should know that you may encounter inconsistencies during the import process.

   - If you packaged existing business objects in step 2, then install these transport files.
   - If you are using non-implemented business objects, then simply install the latest transport file for the business object that you want to use. You must install the correct version-specific transport files.

   For more information on installing these transport files, see "Connector transport file installation" on page 47.
5. Compile all programs and resolve syntax errors associated with the ABAP development.

   The easiest way to find syntax errors is to generate each function group associated with each business object and fix the errors one at time. Repeat this process until all function groups compile successfully. Be sure to generate any other programs, such as triggering programs, that are not associated with a function group.

If you are upgrading to SAP R/3 version 4.x, note that the 4.x ABAP handlers use the product namespace /CWLD/. For special considerations for upgrading to the connector supporting SAP R/3 version 4.x, see "Connector for SAP R/3" on page 118.

6. Apply the event detection mechanisms.

   For user exits, the precise location may be different now. Search for key SAP lines of code to make a best approximation.

7. Test the new environment and make modifications as needed.

   Only a full system test enables you to work out any problems with the upgrade. Test your event detection mechanisms by running the appropriate transaction or program and sending business objects to the SAP system. Use the connector's log within the SAP system to help identify other issues.

   For information on testing business objects, seeChapter 8, "Testing a business object for the ABAP Extension module," on page 101.

Upgrade considerations

The following sections provide reference information for the upgrade scenarios. This reference information is provided to help with the upgrade process for the connector for SAP R/3 version 4.6 and IDocs.

Connector for SAP R/3

The connector for SAP R/3 version 4.x uses the product namespace /CWLD/; the following guidelines facilitate the effort to make your ABAP handlers work in this renamed environment. See Chapter 5, "Business object processing in the ABAP Extension module," on page 57 for more information on how objects are processed and for background information for developing objects.

Business objects that use dynamic retrieve or dynamic transaction

The functionality for converting transaction-based (Dynamic Retrieve and Dynamic Transaction) type business objects is provided through the new IBM WebSphere InterChange Server Station. The business object can be downloaded to a text file from transactions YXDY (Dynamic Retrieve) and YXTD (Dynamic Transaction) in the old system and then uploaded to the new tables using IBM WebSphere InterChange Server Station in the new system. Do this from the Tools tab using the Object MetaData option.

Keep the following in mind:

- The Long Text Declarations for transaction-based Dynamic Retrieve need to be manually entered into the new table.
- The Table Declarations for transaction based Dynamic Retrieve need to be manually ported from the old include program to the new tables declaration include program.

Business objects that use IDoc or BAPI handlers, and custom work

You must redirect SAP R/3 version 3.x business objects that begin with Y* to the product /CWLD/ namespace. Only the names have changed. SAP's "where used list" functionality greatly facilitates the search for all of the references that need to be changed. Following is a list of the most common references that need to be changed. Test to ensure your search is complete.

Table 18 shows the changes for the /CWLD/ namespace naming convention. The parameter lists do not require changes.

Table 18. Namespace object name changes

Old name	New name
Interface parameters of the function modules	
YXR_EVENT-OBJ_KEY	/CWLD/LOG_HEADER-OBJ_KEY (in three places)
YXR_LOG_H-LOG_NR	/CWLD/LOG_HEADER-LOG_NR
YXR_RFCRC-YXR_RFCRC	/CWLD/RFCRC_STRU-RFCRC
Changes normally in the TOP include of the business object function group	
YXR_CNST	/CWLD/CONSTANTS
YXRIFRM0	/CWLD/INBIDOC_FRMS0
Data elements	
YXR_VERB	/CWLD/OBJ_VERB
Table structures	
YXR_CONFIG	/CWLD/CONF_VAL
YXR_EVENTS	/CWLD/EVT_CUR
YXR_LOG_I	/CWLD/LOG_ITEM
YXR_RFC_S	/CWLD/OBJ_STRU
Program referenced in the LOG_UPDATE perform statement	
SAPLYXR1	/CWLD/SAPLLOG
Triggering function modules (the parameter lists do not require changes)	
Y_XR_COMMIT_IDOC_RAISE_DELETE	/CWLD/ COMMIT_IDOC_RAISE_DELETE
Y_XR_/ADD_TO_QUEUE	/CWLD/ADD_TO_QUEUE

Additional IBM ABAP components

In addition to upgrading the custom objects and custom work, you must:

- Upgrade any ABAP code from the old event restriction program to the new event restriction program.
- Manually upgrade all configuration objects and configuration values from the old tables to the new tables.
- Upgrade any event distribution entries from the old table to the new table.
- Upgrade the log object links from the old table to the new table.

Give special consideration to production sites that already have events in the existing SAP R/3 version 4.x event tables. The transfer of these events from the existing event table to the new event table should be coordinated with IBM Technical Support.

Packaging and re-creating IDocs

This section applies to IBM WebSphere SAP R/3 version 3.x objects only.

Because you cannot transport IDoc objects from SAP R/3 version 3.x, you must manually re-create them in the new SAP R/3 system. To do this, you need to:

- Capture the IDoc structure and segment definitions
- Manually re-create the IDocs

Capture the IDoc structure and segment definitions

To capture the most useful representation of an IDoc, capture the overall structure that identifies all of the segments, and then capture business object definitions for each segment. Once you have a clear representation of the IDoc, you can us it to manually re-create it in the new system.

If you have access to the old and new systems, you can simply copy and paste the business objects between the systems. However, if both systems are not available, then you should record the IDoc and segment definitions outside of the SAP system for reference. Although this is optional, it is strongly recommended that you record the definitions.

To download the most useful representations of the IDocs and the segment definitions, first download the overall structure of the IDoc, and then download the IDoc segment definitions.

Downloading the overall IDoc structure: To download the overall IDoc structure:

1. Go to the Develop IDocs Type screen (transaction WE30).
2. Enter an IDoc object name, and then click Display (F7).
3. Expand the IDoc structure so that all segments are visible.
   a. Download a text version of the structure.
   b. From the System menu, click List, click Save, and then click Local File.
   c. Accept the default option unconverted, and then click Enter.
      The file is downloaded as a text file and can be viewed in any text editor.
   d. Specify the location to download the file, and then click Transfer.

Downloading the segment definitions: You can download only one segment definition at a time. Repeat the following steps for each segment. To download a segment definition:

1. Go to transaction SE11 and enter the segment name.
2. From the Dictionary Object menu, click Print.
   Make sure the Table Structure box is checked.
3. Deselect the Print immediately checkbox, check the new spool request check box, and then click Continue.
4. Go to the Spool Request Selection screen (transaction SP01) to view the print request.
5. Click Execute, select the checkbox next to the request, and then click Display comments.
6. Convert the data to a downloadable format.
   a. From the Goto menu, click List Display.
   b. Download a text version of the segment. From the System menu, point to List, point to Save, and then click Local File.
   c. Accept the default option unconverted, and click Enter.
      The file is downloaded as a text file and can be viewed in any text editor.
   d. Specify the location to download the file, and then click Transfer.

Once you have represented the object using text files, you can import them into a spreadsheet application to set up the object hierarchy. This facilitates the creation of IDoc segments, because you can cut and paste the fields directly into the segment editor in the SAP application.

Manually re-create the IDocs

Once you have a representation of the IDoc, you must manually re-create it in the new system. The SAP R/3 version 4.x environment uses different tables to store IDoc type and segment definitions than does SAP R/3 version 3.x. As a result, you must use SAP's tools to redefine the IDoc definitions to update the proper tables. There are two steps to this process:

- Re-create the segment definitions using the Develop Segments screen (transaction WE31).
- Re-create the IDoc type and assign all of the segments to it.

A common error message encountered when re-creating segments by assigning the SAP R/3 version 3.x data element to the new segment field is `Invalid data element`. SAP replaced many of the SAP R/3 version 3.x data elements with data elements that have an underscore followed by the letter D (_D) at the end of the SAP R/3 version 3.x name. For example, CHARG in SAP R/3 version 3.x is Batch Number for the data element and is replaced in SAP R/3 version 4.x with CHARG_D.

If a data element does not exist in the new form, find a new form in the SAP R/3 version 4.x system. The data element must have the same type and length as the original in SAP R/3 version 3.x system. The description does not affect processing and is visible only in the log.

Attention: Do not rename the IDoc, segments, or segment fields because there is a direct relationship between the IDoc definition and the IBM WebSphere business object repository. In addition, the ABAP functions used to process the IDoc also rely on these names.

Part 3. ALE module

Chapter 11. Overview of the ALE module

This chapter describes the ALE (Application Link Enabling) module of the IBM WebSphere Business Integration Adapter for mySAP.com (SAP R/3 Version 4.x). ALE is part of the integration layer within SAP's business framework. The ALE Module enables businses process integration and asynchronous data communication between two or more SAP R/3 systems or between SAP R/3 and external systems.

This chapter contains the following sections:

Overview of ALE technology

The ALE Module is best used for objects such as batch objects, that are asynchronous in nature. It uses push technology that requires that there be a server listening for events. Processes called registering and installing notify the server what to listen to and from whom to expect information. Registering involves using a program itendifier to give the SAP Gateway a communication point with listener threads (servers). Function module definitions within the server interpret data that is pushed out of SAP by providing a template for this data.

The ALE module uses the RFC Server module for event handling. The ALE module uses MQ Series queues for Transaction ID (TID)and IDocs management. The connector checks for subscriptions when processing the data from SAP to the connector, resulting in transactions remaining in SAP until the collaboration is started.

- The integration broker sends a WebSphere Business Integration Adapter business object for SAP. The business object's data represents a processing request to the connector. The connector converts the business object to a table format compatible with the SAP Intermediate Document (IDoc) format. The connector uses Remote Function Calls (RFCs) to the ALE interface to pass the IDoc data to the SAP R/3 system.
- The connector receives data representing an application event from SAP in IDoc table format. It converts the data to a WebSphere Business Integration Adapter business object for SAP before sending it to the integration broker. The connector uses RFCs to the ALE Module to receive the data from the ALE interface.

Important: In releases of the connector prior to version 4.8.2, the connector used collaborations, business objects, and maps to store Transaction IDs (TIDs) and their status in the repository, and used the local file system to store IDoc data. Version 4.8.2 of the connector replaces the previous management of TIDs and IDoc data with the use of MQSeries queues.

Note: Because the ALE Module uses asynchronous communication, it cannot be used when cross-referencing is required.

ALE Module components

The ALE Module is written in Java and extends the vision connector framework. The module consists of:

- Connector framework
- Connector's application-specific component for ALE
- Two ALE business object handler classes (one for event processing and one for request processing)
- SAP RFC libraries
- SAP SAPJCo connector
- Application-specific component for the RFC Server (used for event processing only).

  The ALE Module uses the RFC Server connector component because the similarities for event processing both support RFC calls directly from the SAP application.

SAP delivers the RFC libraries in Java and C. The connector is delivered and run as a Java archive (JAR) file.

Figure 16 illustrates the architecture of the ALE Module.

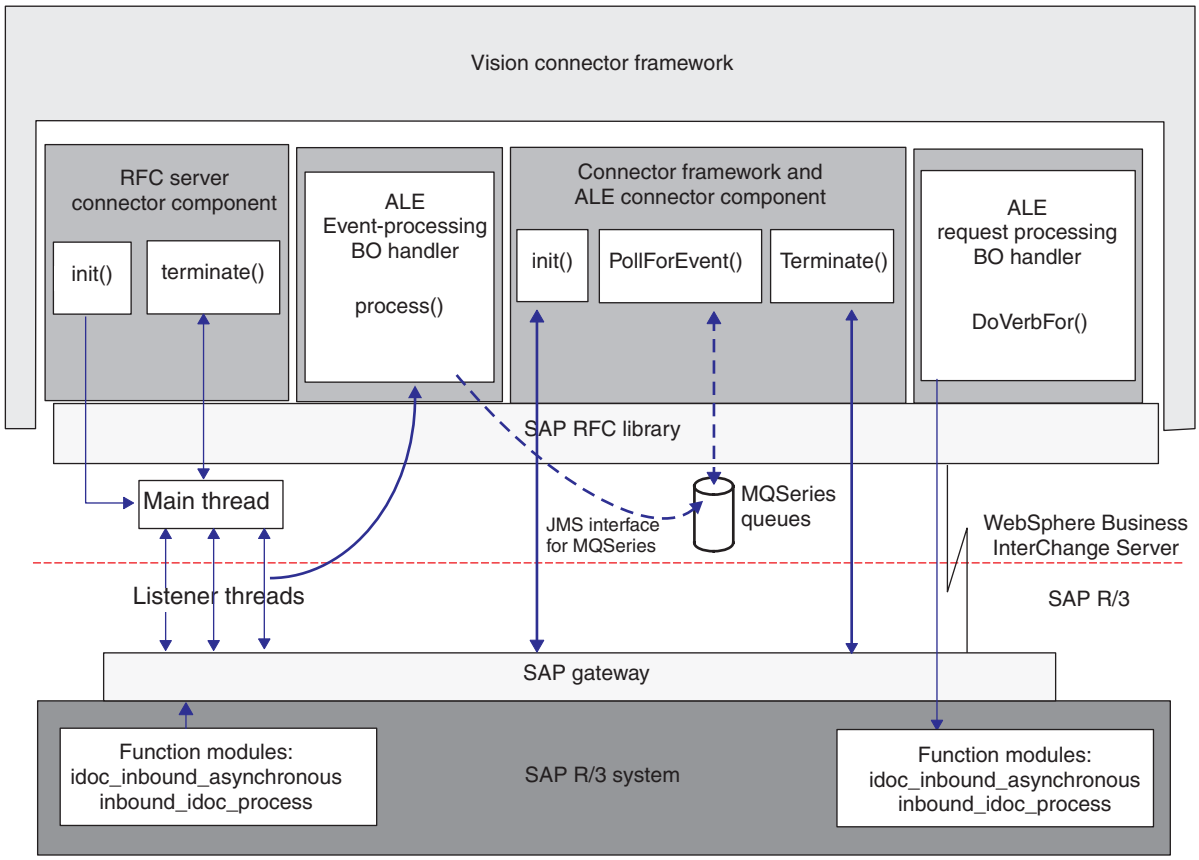


Figure 16. ALE Module architecture

Event processing components

When processing events from SAP, the connector uses the components illustrated in Figure 16 on page 126 in the following ways:

- The vision connector framework starts the RFC Server connector component, which spawns listener threads. Each listener thread uses the RFC library and the SAP gateway to register a single handle to the SAP application.
- The listener thread processes events from the SAP application.

  An **event** is the execution of an ABAP function that transfers data to the listener. The event data sent by SAP may represent one or more such heterogeneous executions.

  Each event from SAP is considered a transaction. The connector uses a two-step process with a Transaction ID (TID) to handle each event, guaranteeing once-only delivery of data from SAP to the connector.
- MQSeries queues persistently store a JMS-MQ message for each event. Each JMS-MQ message stores the TID identifying the event, the status of the TID, the IDoc data associated with the event, and the processing status of the IDoc.
- The connector's polling process creates WebSphere business objects from the stored event message, and sends the business objects to the integration broker.
- The business integration system tracks unprocessed events to handle their recovery in case the integration broker or the connector goes down. When the integration broker or the connector is restored, the connector automatically resubmits these events.

Request processing components

When processing requests from the integration broker, the connector uses the components illustrated in Figure 16 in the following ways:

- The ALE Module uses the SAP RFC library and the SAP Gateway to open an RFC connection to the SAP R/3 application.
- The ALE request-processing business object handler processes requests from the integration broker, converting them from business object format to IDoc data based on the SAP IDoc format:
- For every request sent to the application, the ALE Module persistently stores Transaction IDs (TIDs) in a TID queue in a JMS-MQ message. The TID guarantees that the request is delivered once and only once. However, if the intergration broker sends an object that has the same value in the transaction ID attribute, this object will be processed again. Once an object has been sucessfully sent the expectation is that the integration broker will not send the object again.
- The ALE Module releases the connection to the SAP R/3 application.

Listener threads

Listener threads handle all of the ALE-specific RFC calls between the ALE Module and the SAP application. When the connector starts up, the `init()` method of the RFC Server Module creates a main thread that spawns a configurable number of listener threads. Each listener thread opens a handle to the SAP Gateway.

The listener threads do the following:

- Use a program identifier to register with the SAP Gateway.
- Identify to the SAP Gateway the ALE-specific RFC-enabled functions that they support. These functions are `idoc_inbound_asynchronous` and `inbound_idoc_process`.
- Receive events from the ALE-specific function.

- Instantiate the event-processing ALE business object handler.

A thread listens continuously in a synchronous manner for events from the ALE-specific functions that it supports.

Transaction IDs

SAP uses a transaction and its corresponding ID to frame an event, guaranteeing that each piece of data is delivered once and only once from SAP. SAP sends a Transaction ID (TID) with the event data. To manage the TIDs centrally for event and request processing, the connector stores each TID as a JMS-MQ message on an MQSeries queue. When processing events, it also stores the associated IDoc data as the message body. The connector stores the TID, TID status, and the IDoc's processing status in the message header.

ALE-specific business object handlers

Two ALE-specific business object handlers are provided, one for event processing and one for request processing.

Event-processing business object handler

A listener thread instantiates the event-processing business object handler, which does the following:

- Retrieves the RFC event data from SAP.
- Creates a JMS-MQ message to persistently store and manage the transaction ID that SAP sends with the event.
- Stores the data of the one or more IDocs received from SAP in the JMS-MQ message.
- Returns a response to the ALE-specific function through the SAP Gateway. The response indicates that the transaction has been completed.

Request-processing business object handler

The vision connector framework instantiates the ALE request-processing business object handler, which checks for a value in the TransactionId attribute in the WebSphere business object for SAP. If this value exists, it continues with the following steps.

1. Obtains a TID either from the JMS-MQ message or from SAP.
2. Converts the business object data to the IDoc data format defined by the desired function module interface for the RFC call into SAP.
3. Makes the RFC call to the ALE interface.
4. Updates the status of the TID for this request in the JMS-MQ message.
5. Returns a success response to the integration broker.

Structure of the business object for SAP

A WebSphere business object for SAP represents each IDoc as a parent wrapper business object that contains two child business objects: a control record business object and a data record business object. The control record business object contains the metadata required by the connector to process the business object. The data record business object contains the actual business object data to be processed by the SAP application, and the metadata required for the connector to convert it to an IDoc structure for the RFC call.

The adapterIBM WebSphere Business Integration Adapter for mySAP.com includes a business object definition for the control record. The definition file,

BO_SAPIDocControl.txt, is located in the \repository\SAP directory. The ALE Module uses the same business object definition for 3.X and 4.X versions of SAP.

The TABNAM attribute in the control record business object indicates which SAP function module the parent wrapper business object calls:

- A value of EDI_DC40 indicates the idoc_inbound_asynchronous function module, which the connector uses only for SAP 4x.
- A value of EDI_DC indicates the inbound_idoc_process function module, which is provided for backward compatibility with SAP 3x.

In addition, the following attributes must have values for SAP to properly process the object in ALE. These values are based on your ALE configuration:

- Name_of_table_structure
- Client
- Name_of_basic_type
- Logical_message_type
- Partner_type_of_sender
- Partner_number_of_sender
- Partner_type_of_recipient
- Partner_number_of_recipient

The DOCNUM attribute in both business objects establishes the relationship between the data record business object and the control record business object.

When processing service call requests, the ALE Module can handle multiple IDocs in a single business object. Before it can do so, however, you must add another multiple-IDoc wrapper business object around two or more parent wrapper business objects. This top-level multiple IDoc wrapper business object contains an attribute that represents an array of parent wrapper business objects. For more information, see “Parent wrapper business object” on page 149.

The adapterIBM WebSphere Business Integration Adapter for mySAP.com includes a business object generation tool, SAPODA. This tool uses an IDoc definition text file to generate business object definitions for the ALE Module. For more information on developing business objects for the ALE Module, see Chapter 13, “Developing business objects for the ALE module,” on page 145 and Appendix E, “Generating business object definitions using SAPODA,” on page 291.

Chapter 12. Configuring the ALE module

This chapter describes the configuration and use of the ALE Module. The connector component of the Adapter Guide for mySAP.com (R/3 V.4.x) should be installed before performing the configuration tasks described in this chapter.

For more information on installing the connector, see Chapter 2, "Installing and configuring the connector," on page 11.

This chapter contains the following sections:

Prerequisites to running the ALE Module

To enable the connector to store the TID and IDoc data persistently during event processing, and to store the TIDs persistently during request processing, you must do the following:

- Verify that the following are installed and running on your system:
  - MQSeries (not included)
  - TCP/IP
- For event processing, create the following MQSeries queues, whose names are specified by the corresponding connector-specific configuration properties:
  - Archive (`SAPALE_Archive_Queue` property)
  - Event (`SAPALE_Event_Queue` property)
  - Work-in-Progress (WIP) (`SAPALE_Wip_Queue` property)
  - Errors (`SAPALE_Error_Queue` property)
  - Unsubscribed (`SAPALE_UnSubscribed_Queue` property)

  For information about how the connector uses these queues, see "Running the ALE Module" on page 134.
- For request processing, create a single MQSeries queue whose name is specified by the `SAPtid_Queue` configuration property.

  For information about how the connector uses this queue, see "Running the ALE Module" on page 134.
- To use the ALE Module to process large IDocs or IDoc Packets:
  - Increase the Maximum Message Length of the MQSeries Queue Manager and its queues. This length defaults to 4194304 bytes
  - Increase the log file size and the number of log files when you create the Queue Manager
  - If Channels are used for the MQ Series Queue Manager, then increase the Maximum Message Length of the channel

    Refer to the MQSeries System Administration publication for more information on configuring the log files.

ALE Module directories and files

Table 19 lists the directories and files used by the ALE Module.

Table 19. ALE Module directories and files

Filename	Events	Requests	Description
BO_SAPIDocControl.txt	Yes	Yes	Control record business object definition file. Located in the \repository\SAP directory.
EventState.log file	Yes	No	Located in the directory specified in the AleEventDir configuration property, the connector logs information to this file about successfully processed IDocs in a JMS-MQ event message. Note: The connector does not create the log file automatically the first time it processes an event. You must create this file for before you run the connector for the first time.

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.

Configuring the ALE Module

Before you can use the ALE Module, you must:

- Add the module name for the ALE Module to the module's property. The module name is ALE.
- To enable event-processing with TID management, you must configure the appropriate connector-specific properties.
- To cause the connector to update a standard SAP status code after the ALE Module has retrieved an IDoc for event processing, configure the specific properties and inbound parameters of the Partner Profile of the Logical System in SAP to receive the ALEAUD message type. For more information and a full listing of relevant properties, see "Configuring SAP To update IDoc status" on page 133.
- Set the remaining required standard and connector-specific configuration properties.

  To set the connector configuration properties, use Connector Designer. For more information on setting the connector configuration properties, see "Configuring the connector" on page 17 and Appendix B, "Standard configuration properties for connectors," on page 241.

Important: Connector polling is required for this module to manage errors properly when it processes application events. Therefore, do not set the value of the connector's PollFrequency property to key or to no. Do not allow the SAP application to trigger events to the connector until you have verified that the connector's log displays the installation of the required RFC functions.

Checking the SAP configuration

Before running the ALE Module, verify that the SAP system is properly configured to process business objects:

- Check that the logical systems are defined and assigned for the SAP system and external system (transaction code SALE).
- Check that the distribution model has been maintained, and that the required message types have been added to the model (transaction code BD64).
- Check that there are partner profiles for the logical system or distribution model (transaction code WE20).

Checking MQ configuration

Verify that message queues are properly configured.

For event processing:

- Check that the SAP application (transaction code SM59) matches the program ID specified in the `RrcProgrmId` configuration property. For more information on setting up a TCP/IP port see "Registering the RFC Server Module with the SAP gateway" on page 189..
- Check that the WIP (`SAPALE_Wip_Queue`), Event (`SAPALE_Event_Queue`), Error (`SAPALE_Error_Queue`), Unsubscribed(`SAPALE_Unsubscribed_Queue`), and Archive queues (`SAPALE_Archive_Queue`) are defined and running in MQSeries.

For request processing, check that the request queue (`SAPtid_Queue`) is defined and running in MQSeries.

Configuring SAP To update IDoc status

To cause the connector to update a standard SAP status code after the ALE Module has retrieved an IDoc for event processing:

- Set the `AleUpdateStatus` configuration property to `true` and set values for the `AleSuccessCode` and `AleFailureCode` configuration properties.
- Configure the inbound parameters of the Partner Profile of the Logical System in SAP to receive the `ALEAUD` message type.

For more information, see "Updating the IDoc status in SAP" on page 138.

Configuring SAP

Configure the inbound parameters of the partner profile of the logical system to receive the `ALEAUD` message type. Set the following properties to the specified values:

Table 20. Configuring SAP to receive IDoc status

SAP Property	Value
Basic Type	`ALEAUD01`
Logical Message Type	`ALEAUD`
Function module	`IDOC_INPUT_ALEAUD`
Process Code	`AUD1`

Setting connector-specific configuration properties

Set the following required connector-specific configuration properties to return IDoc status:

- "AleUpdateStatus" on page 20
- "AleSuccessCode" on page 21
- "AleFailureCode" on page 21

Set the following required connector-specific configuration property to process events and requests: "SAPtid_QueueManager" on page 25

You may also set the following optional connector-specific configuration properties:

- "AleSelectiveUpdate" on page 20
- "AleStatusMsgCode" on page 21
- "AleSuccessText" on page 21
- "AleFailureText" on page 21

Connecting to remote queue managers

Set the following required connector-specific configuration properties for remote queue managers:

- "SAPtid_MQChannel" on page 24
- "SAPtid_MQPort" on page 24
- "SAPtid_QueueManager" on page 25
- "SAPtid_QueueManagerHost" on page 25
- "SAPtid_QueueManagerLogin" on page 25
- "SAPtid_QueueManagerPassword" on page 25

Running the ALE Module

When processing application events, the ALE Module receives events that the SAP application pushes to the connector. When processing requests, the ALE Module receives business object requests from the integration broker and sends them to the SAP application.

Initialization and termination

The `init()` method opens an RFC connection to the SAP R/3 application through the SAP Gateway. If the connector fails to initialize, it terminates the connection using the `terminate()` method. The connector terminates by disconnecting from the SAP Gateway.

When processing application events or business object requests, the connector's initialization process performs the following tasks:

1. Registers with the SAP Gateway the Program ID specified in the `RfcProgramID` connector configuration property. For information on setting the Program ID as a TCP/IP port see "Registering the RFC Server Module with the SAP gateway" on page 189..
2. Opens an MQSeries session to the queues configured for the connector.
3. Verifies that the required MQSeries queues for event and request processing have been created. If they have not been created, the process terminates the connector.

Because the connector supports multi-threading, when the ALE Module processes requests from the integration broker, it uses SAP's Java Connector (SAPJCo) connection pool of such handles.

Important: When you use the ALE module to process application events, connector polling is required to properly initialize the module (to install the RFC functions on the server), and for it to properly manage errors. Therefore, do not set the value of the PollFrequency property to `key` or to `no`. Do not allow the SAP application to trigger events to the connector until you have verified that the connector's log displays the installation of the required RFC functions.

Processing business objects

The ALE Module's processing of WebSphere business objects for SAP is initiated either through event processing or request processing.

When business object data is returned from SAP's Java Connector (SAPJCo) API, the ALE Module receives values for DATS and TIMS fields in the following formats: `YYYY-MM-DD` (the hyphens are included) for the DATS data element, and `HH:mm:ss` (the colons are included) for the TIMS data element. The capitalized `HH` denotes 24-hour time, and not 12-hour time. When processing events, the ALE Module changes these formats to fit the 8-character and 6-character maximum size of their corresponding business object attributes. The connector shortens the length of the value by removing the hyphens from the date data and the colons from the time data.

Event processing

Two RFC-enabled functions in an SAP application initiate all event processing for the ALE Module. The ALE's business object handler for event processing supports the functions `idoc_inbound_asynchronous` and `inbound_idoc_process`.

When processing events, this business object handler persistently stores business objects in an MQSeries queue. The connector maintains the Transaction IDs (TIDs) associated with the RFC call to guarantee that each piece of data is delivered once and only once.

Important: A single RFC call can send the data for one or more IDocs. Therefore, an MQSeries queue may contain a JMS-MQ message that represents multiple IDocs, each of which represents a business object. Each RFC call is associated with a single TID.

Processing events in the MQSeries queue: Figure 17 on page 136 illustrates how the ALE Module processes the MQSeries queue.

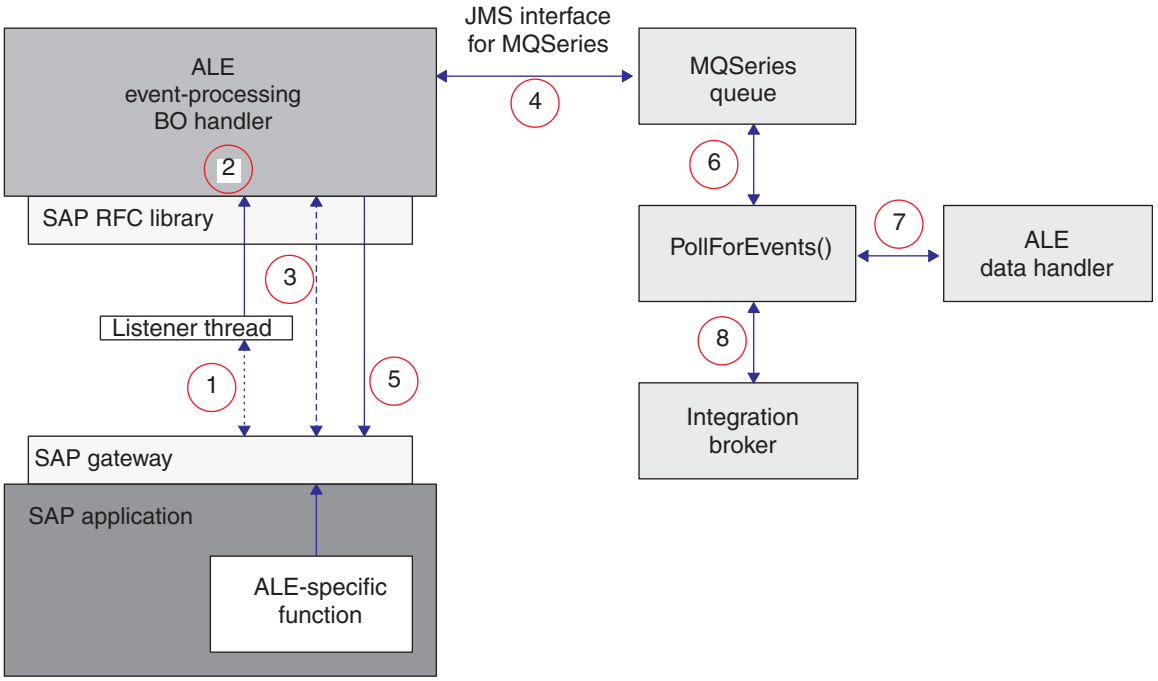


Figure 17. Business object event processing

Business-object event processing for the ALE Module executes in the following manner:

1. An RFC function pushes event data to the SAP Gateway, where a listener thread picks up events. The thread checks the TID associated with the event to determine whether a JMS-MQ message exists for the TID:
   - If the TID has not been sent previously, the connector continues to 2.
   - If the TID has been sent previously, the connector's behavior depends on the state of the previous transaction. If TidStatus is `CREATED`, the connector removes the IDoc data from the message. If the status is `ROLLBACK`, the connector changes the status to `CREATED`, and if IDoc data exists in the message, the connector removes the IDoc data from the message. If the status is `EXECUTED`, the connector returns control to SAP.
2. The listener thread instantiates the ALE event-processing business object handler, which retrieves the RFC-interface data from the SAP Gateway.
3. The business object handler formats each transaction into a JMS-MQ message, which it stores persistently in the queue specified by the `SAPALE_Wip_Queue` configuration property.

   Each JMS-MQ message represents a single RFC call. Each RFC call can represent one or more business objects associated with a single TID. The connector stores the TID in the message's CorrelationID property, sets the TidStatus to `CREATED`, and sets the IDocProcessStatus to `unknown`. The connector uses the message body to store IDoc data.
4. After each transaction completes, the connector changes the value of TidStatus and sends a confirmation back to SAP indicating that the transaction is complete. After SAP receives the confirmation, it removes the TID and its associated data from the SAP application.

   If the AleUpdateStatus configuration property evaluates to `true`, the connector updates the status of the IDoc in SAP. If it retrieves a packet of IDocs, it updates the status of all IDocs in the packet. For more information, see "Updating the IDoc status in SAP" on page 138.

5. The connector moves the JMS-MQ message from the WIP queue to the queue specified by the SAPALE_Event_Queue configuration property.
6. The ALE Module's polling thread picks up the event message from the Event queue.
7. The connector instantiates an ALE data handler that will convert the data in the message body to business objects for posting to the integration broker.
8. The connector attempts to post each business object to the integration broker. If the integration broker is WebSphere Interchange Server, the connector first checks if there are subscriptions for the business object. After processing all the business objects in the message body, the message's IDocProcessingStatus and BOProcessingStatus are updated and the message is moved to the queue specified by the SAPALE_Archive_Queue property. For more information on IDocProcessingStatus see, ″Creating archive messages″ and on BOProcessingStatus see, ″Structure of JMS-MQ message for event and archive processing″.

The ALE module uses FIFO (First In, First Out) to maintain the processing order when reading the messages from the Event queue.

Important: Connector polling is required for this module to manage errors properly when it processes application events. Therefore, do not set the value of the connector's PollFrequency property to key or to no. Do not allow the SAP application to trigger events to the connector until you have verified that the connector's log displays the installation of the required RFC functions.

Structure of JMS-MQ message for event and archive processing: Table 21 describes the structure of the message that the connector sends to the Event and Archive queues.

Table 21. Structure of JMS-MQ message for event and archive processing

JMS Message Header Property	Description
CorrelationId	The connector sets the value of this property from the Transaction ID (TID) sent by SAP.
TidStatus	Maintains the status of the TID.
IDocProcessStatus	Maintains the status of the IDoc object during event processing.
BOProcessingStatus	Maintains the status of all IDocs in the message using the format, <CID> :: <IDoc sequence number><Status symbol>. Possible status symbols are S for Success, F for fail and U for unsubscribed. For example ″<CID> :: 0S, 1F, 2U″ means the first IDoc was successful, second failed, and third was unsubscribed for CorrelationId = <CID>.

Table 22 describes the possible values for the IDocProcessStatus property after an event is moved to the Archive queue.

Table 22. Archive queue values for the IDocProcessStatus message property

IDocProcessStatus property value	Event status	Description
success	Success	All business objects in the message have been posted with no errors.

Table 22. Archive queue values for the IDocProcessStatus message property (continued)

IDocProcessStatus property value	Event status	Description
`partial`	Partial success	One or more but not all business objects in the message have been posted with an error. If the integration broker is WebSphere Interchange Server, one or more but not all business objects in the message have been posted with an error or are unsubscribed.
`unsubscribed`	Unsubscribed	If the integration broker is WebSphere Interchange Server, all business objects in the message are unsubscribed.
`fail`	Fail	All business objects in the message have been posted with an error.

Creating archive messages: When the message is moved from the Event queue to the Archive queue, the IDocProcessingStatus and BOProcessingStatus are updated. The message body remains unchanged.

For example, assume the connector processes an event message with four IDocs, each of which it transforms or attempts to transform into a business object, with the results illustrated in Table 23:

Table 23. Archive message creation

Status of IDoc or business object	Resulting archive message
Successfully transforms the first IDoc, and posts the business object to the integration broker	The IDocProcessStatus is updated to success and the BOProcessingStatus is `<CID> :: 0S`
Fails to transform the second IDoc into a business object	The IDocProcessStatus is updated to partial and the BOProcessingStatus is `<CID> :: 0S, 1F`
Successfully transforms the third IDoc, and posts the business object to the integration broker	The IDocProcessStatus is set to partial and the BOProcessingStatus is `<CID> :: 0S, 1F, 2S`
Successfully transforms the fourth IDoc, but the business object created is not subscribed in the integration broker	• The IDocProcessStatus is set to partial and the BOProcessingStatus is `<CID> :: 0S, 1F, 2S, 3U` • After processing the last IDoc, moves the message from the Event queue to the Archive queue and gives it IDocProcessStatus of partial and BOProcessingStatus of `<CID> :: 0S, 1F, 2S, 3U`

Updating the IDoc status in SAP: To cause the connector to update a standard SAP status code after the ALE Module has retrieved an IDoc for event processing, you must:

- Set the `AleUpdateStatus` configuration property to `true` and set the value of the `AleSuccessCode` and `AleFailureCode` configuration properties.
- Configure the inbound parameters of the Partner Profile of the Logical System in SAP to receive the `ALEAUD` message type.

If AleUpdateStatus evaluates to true, the connector sends the ALEAUD IDoc to SAP with status code information and descriptive text. The ALEAUD IDoc calls the IDOC_INPUT_ALEAUD function module. The connector supports sending the following status codes to this function module:

- IDoc has been completely posted in the business integration system.

  The AleSuccessCode connector-specific configuration property can have a value of 52 or 53. SAP converts this value to 41.
- IDoc cannot be processed in the business integration system.

  The AleFailureCode connector-specific configuration property can have a value of 68. SAP converts this value to 40.

In both of the cases above, the business integration system does not send further status codes that would indicate further processing.

For information on setting the connector-specific configuration properties that are required to return IDoc status, see:

- "AleUpdateStatus" on page 20
- "AleSuccessCode" on page 21
- "AleFailureCode" on page 21

For information on setting the connector-specific configuration properties that are optional to return IDoc status, see:

- "AleSelectiveUpdate" on page 20
- "AleStatusMsgCode" on page 21
- "AleSuccessText" on page 21
- "AleFailureText" on page 21

ALE Module Queue Management utility for event processing

Use this command-line utility for maintenance of MQ queues used by the WebSphere Business Integration adapter for mySAP.com's (v. 5.3.2) ALE module. The utility resubmits event messages, dumps event messages to a file system for viewing, and archives messages to a file system.

An IDoc is processed in a unit of work called a transaction. An SAP transaction containing more than one IDoc is called a transaction packet. The adapter processes transactions and transaction packets by using an MQ message to hold the IDoc or IDocs. The adapter converts the IDoc into its corresponding business object. The ALE module handles processing of IDocs in a two-step process: SAP to the adapter, then the adapter to the broker. Exceptions are handled differently for each step.

For more information about MQ messages, see the WebSphere Business Integration Library: http://www.ibm.com/software/integration/wmq/library/.

Processing IDocs from SAP to the adapter: If the adapter detects unsubcribed or unsupported business objects or raises any exceptions during IDoc transmission, the adapter will fail the SAP transaction. Failed transactions can be viewed and resubmitted from SAP transaction SM58. Before resubmitting the transaction, address the exception:

- Unsupported: add agent support for the business object.
- Unsubscribed: restart the collaboration for the business object.
- Other exceptions: view the adapter logs to determine the exception and make the necessary correction.

Once this step executes successfully, the transaction with SAP is complete.

Important: To prevent duplicate event delivery, do not resubmit a corrected IDoc transaction or individual IDoc within a transaction packet.

Processing IDocs from the adapter to the broker: If an MQ message contains a single business object and is unsubscribed, the MQ message will be moved to the unsubscribed queue. Each unsubscribed business object within a transaction packet will persist as its own MQ message on the unsubscribed queue. The original MQ message remains intact and contains the processing status of the individual IDocs. Once the transaction packet for the MQ message is completely processed, it is moved to the archive queue.

Before resubmitting the transaction, address the exception:

- `Unsubscribed`: restart the collaboration for the business object.
- Other exceptions: view the adapter logs to determine the exception and make the necessary correction.

After you complete the correction, use the command utility `AleEventUtil` to move the MQ message back to the event queue, resubmitting the event.

When an IDoc contains malformed data or contains 'nodata', the IDoc is moved to the Error Queue as its own message.

Installing and configuring the ALE Module Queue utility: The ALE Module Queue utility is packaged with the SAP adapter. When installed, it has the following directory structure:

```
\Connectors\SAP\Utilities\BIA_AleEventUtil.jar

\Connectors\SAP\Utilities\BIA_AleEventUtil.bat

\Connectors\SAP\Utilities\BIA_AleEventUtil_readme.txt
```

Modify the start script file, `BIA_AleEventUtil.bat`, to capture the following parameters. To access local queue managers, you need only configure `MQQueueManager`.

Variable	Description	Comments
`MQQueueManager`	Name of the Queue Manager	Required parameter.
`MQChannel`	Server connection channel name	Required for accessing remote queue manager.
`MQPort`	Port on which the channel is listening	Required for accessing remote queue manager.
`MQHost`	Host name or IP address on which the Queue Manager is running	Required for accessing remote queue manager.
`MQUser`	Valid username on MQHost	Required for accessing remote queue manager.
`MQPassword`	User password	Required for accessing remote queue manager. Value not encrypted.

Running the MQ management utility: After installing and configuring the utility, navigate to the directory where the ALE Module Queue Management Utility is installed. Valid commands for the utility are:

`-c <choice>` (valid options are [`move, archive, dump, replicate`])

`-i <inputq>`

`-o <outputq>`

`-f <outputfile>`

`-d <date>`

`-u <unique message ID>`

`-n <replication count>`

Note: When there is an existing file with the same name, the `archive` command will raise an exception but the `dump` command will overwrite the file.

To dump the contents of a message to a file, from a command prompt change to the directory where the utility is installed and run the following command:

```
BIA_AleEventUtil -cdump -i<QueueName> -f<OutputFileName>
```

To move messages from one queue to another, run the following command. This command will move all the messages in the queue:

```
BIA_AleEventUtil -cmove -i<FromQueue> -o<ToQueue>
```

To move a single message, use the additional paramater of `MessageIdByte` corresponding to the message ID of the desired message:

```
BIA_AleEventUtil -cmove -i<FromQueue> -o<ToQueue> -u<MessageIdByte>
```

To move all the messages equal to or earlier than a specified date, add the `Date` parameter:

```
BIA_AleEventUtil -cmove -i<FromQueue> -o<ToQueue> -d<date(YYYYMMDD)>
```

To archive messages from a queue to a file, removing all messages equal to or earlier than a specified date, use this command:

```
BIA_AleEventUtil -carchive -i<QueueName> -f<ArchiveFileName>
-d<date(YYYYMMDD)>

BIA_AleEventUtil -creplicate -i<QueueName> -n<Replication_count>
-u<messageID> -t<Testing>
```

Request processing

The vision connector framework uses the value of the verb `AppSpecificInfo` property of the top-level business object to instantiate the ALE request-processing business object handler. The `doVerbFor()` method in the request-processing business object handler initiates all business object requests.

The business object handler converts the business object data into two tables that represent the IDoc format and its metadata component, the control record. Once the data is in IDoc format, the business object handler makes an RFC call to the appropriate SAP function module: either `idoc_inbound_asynchronous` or `inbound_idoc_process`. Because ALE is asynchronous, the connector does not wait for a return response.

Important: By default, parent wrapper business objects generated by SAPODA contain a TransactionId attribute. A value in this attribute causes the connector to manage TIDs when processing service call requests. If you do not want TID management for request processing, do not set a value for this attribute. For more information, see "Parent wrapper business object" on page 149.

Note: The value of the TransactionID attribute must be a unique identifier. The value is not the equivalent of a TID in the SAP application. These values are stored in a table within the JMS_MQ message in the queue specified by the `SAPtid_Queue` configuration property. The TransactionID must begin with an alphabetic character.

If the TransactionID attribute does not have a value, the ALE Module sends the request directly to SAP. If the TransactionID attribute has a value, the ALE Module does the following:

1. The connector checks whether the JMS-MQ message in the queue specified by the `SAPtid_Queue` configuration property has this value.
   - If the value of the business object's TransactionID attribute, ObjectID, does not exist in the table of the JMS_MQ message, a new entry is created in the table. ObjectID becomes the key to the table entry. Then the connector retrieves a new TID from SAP and that TID is assigned to this ObjectID. The connector also sets the TidStatus for this ObjectID to `CREATED`
   - If ObjectID does exist in the table, the connector's behavior depends upon the TidStatus for this OjbectID. If TidStatus is `CREATED`, the connector continues to 2. If TidStatus is `ROLLBACK`, the connector changes the value to `CREATED`, and continues to 2. If TidStatus is `EXECUTED`, the key is removed and archived.
2. The connector converts the business object to RFC tables and makes an RFC call to SAP.
   - If the call posts successfully, the connector updates the key's TidStatus to `EXECUTED`.
   - If the call fails to post to SAP or raises an exception, the connector updates the key's TidStatus to `ROLLBACK`.
3. After SAP acknowledges receipt of the RFC call, the connector removes the key from the table, archives the key, and returns a success status to the integration broker.

Archiving: After successfully processing a service call request, the corresponding entry in the table of the JMS-MQ message in the SAPtid_Queue is removed and archived to a directory. A file is created in the \ale\request subdirectory for WINNT or /ale/request for Unix systems. The ale subdirectory is located in the directory where the adapter is started. The entry that has been removed from the table will be used to create the new file. The file name will have the following format: `<ObjectID>_<TID>)_<timestamp>.executed` where ObjectID is the value from the TransactionId attribute, TID is the transaction ID from SAP, and timestamp is the time stamp of when the file was created.

The adapter itself manages the deletion of these archive files using the connector configuration property ArchiveDays. The value in the connector configuration property, ArchiveDays, determines the amount of days these archived files will persist in the ale\request sub-directory. Any files older than the number of days specified in ArchiveDays will be deleted. If this property is not configured, the default value for ArchiveDays is seven days. These archive files can also managed manually by deleting the files yourself.

Resubmission of Failed Requests: For all failed requests indicated by the integration broker, check whether an archive file has been created for the request. If the archive file exists for the Object ID in the request then do not resubmit the request from the integration broker. Resubmit the request if there is no archive file for that ObjectID. Ensure the ArchiveDays connector configuration property is set to a value that will allow for verification of resubmitted requests.

Columns in the table of the JMS MQSeries message for request processing: Table 24 describes the columns of the JMS-MQSeries message that the connector gets from the `SAPtid_Queue`:

Table 24. Columns of JMS-MQ message for request processing

Column name	Description
`ObjectID`	The value that is in the TransactionID attribute of the requested business object. This value is used as the key for the table
`TID`	The transaction ID obtained from SAP
`TidStatus`	Status of the transaction

Chapter 13. Developing business objects for the ALE module

This chapter describes the business objects required for the ALE Module of the IBM WebSphere Business Integration Adapter for mySAP.com (SAP R/3 Version 4.x). It also discusses how the business object generation utility, SAPODA, generates the definitions. The chapter assumes you are familiar with how the connector processes business objects. For more information on the ALE Module, see Chapter 11, "Overview of the ALE module," on page 125.

Use SAPODA to generate business object definitions for this module. SAPODA uses the SAP application's native IDoc (Intermediate Document) definitions as templates for business object definitions for the ALE Module. After creating the definitions, you can use Business Object Designer or a text editor to modify them. You can use SAPODA to generate business object definitions for the ALE Module based upon an IDoc:

- Extracted to a file
- Defined in the SAP system

IDocs must adhere to a specific format for SAP to process them correctly. Therefore, when you develop a business object definition for the ALE Module, ensure that the definition follows the IDoc Structure as defined in SAP.

For information on using SAPODA, see Appendix E, "Generating business object definitions using SAPODA," on page 291.

This chapter contains the following sections:

Creating the IDoc definition file

Before using SAPODA to generate a business object definition from an IDoc definition file, you must create the IDoc definition file for each IDoc you want to support. SAPODA uses this file as input. Use transaction `WE63` in SAP to create the IDoc definition file.

Note: If you use SAPODA to generate the definition from an IDoc defined in the SAP system, you do not need to create this IDoc definition file.

To create the IDoc definition file:

1. In SAP, select transaction WE63 by entering `/oWE63`.
2. Deselect the IDoc record types check box.
3. Select the Basic type field check box.
4. In the Basic type field, enter the basic IDoc type.
5. Select the Output From Segment Fields checkbox.
6. Click the Execute icon at the top of the screen. The IDoc definition is displayed on the screen.

7. Save the definition to a local directory.

Note: If the business object is based upon IDoc extensions, use the Extended Basic Types grouping.

Important: You must log on to the SAP system in English to generate business object definitions from IDoc files. Because SAPODA uses a text field in the IDoc's definition to generate attribute names, and because attribute names must be in English, it is important that you generate definitions from English-language files.

Business object structure

The WebSphere business object for SAP for the ALE Module is made up of a top-level parent wrapper object and two child objects: the control record object and the data record object. This section describes the following:

Illustration of business object structure

Figure 18 illustrates the structure of a WebSphere business object for the ALE Module.

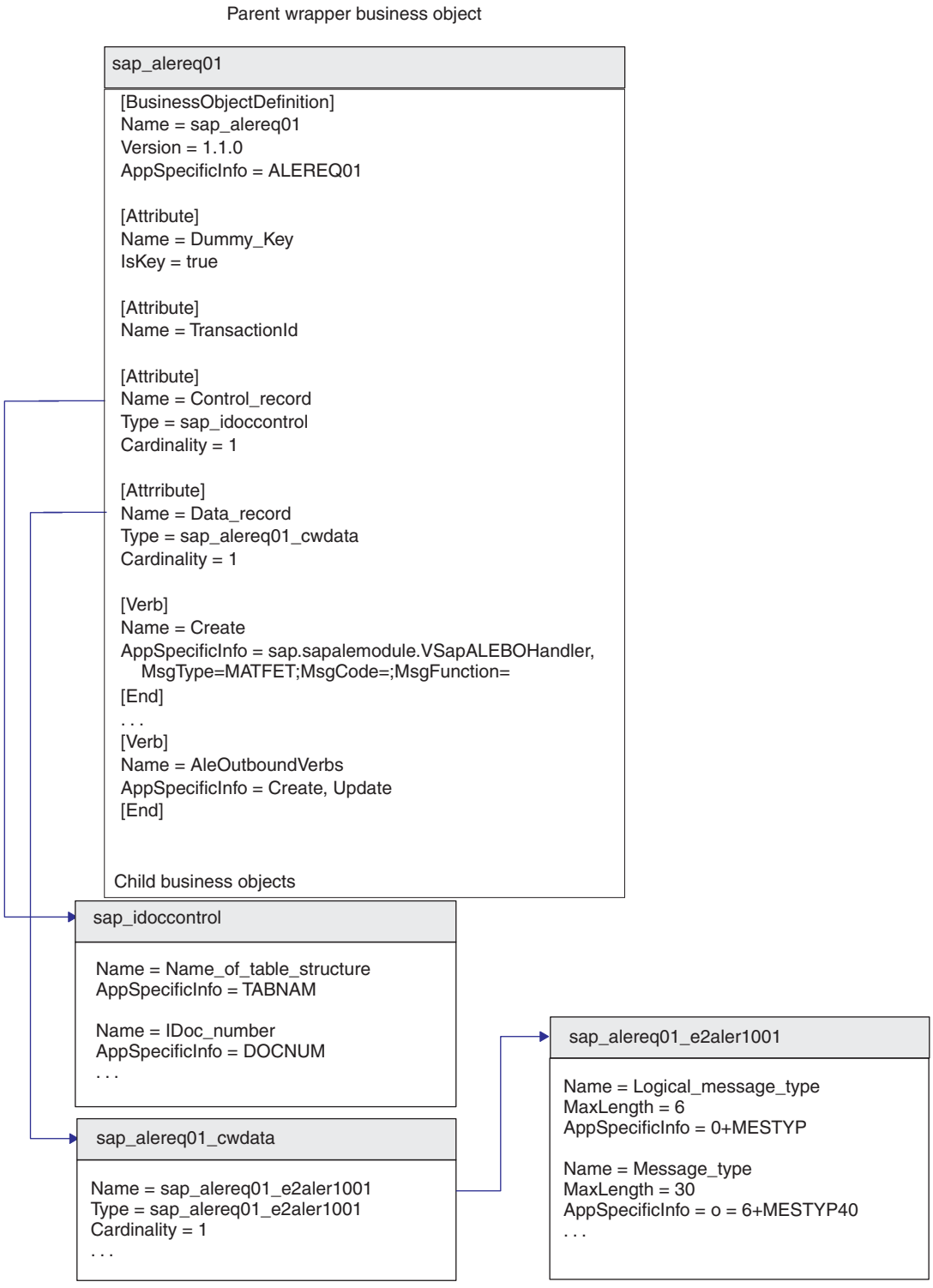


Figure 18. Relationship of WebSphere business objects for SAP and an IDoc

Business object naming conventions

This section describes the following:

- "Standard naming conventions" on page 148

• "Naming conventions for IDoc extensions" on page 149

Standard naming conventions

The ALE Module requires its business objects to follow the naming conventions described in Table 25. SAPODA, which generates all but the control record business object, derives the business object and attribute names from the IDoc definition in accordance with these conventions.

Table 25. IBM WebSphere SAP business object naming conventions

IBM WebSphere business object or attribute	Name	Type
Parent wrapper business object	*BOprefix_BasicIDocType* **Note:** The illustrations in this chapter use SAP_ or sap_ as the business object prefix. You can specify your own meaningful prefix when you create your business object definitions.	n/a
Control Record business object	Control_record	sap_idoccontrol
Data Record business object	Data_record	*BOprefix_BasicIDocType*_cwdata
Data Record child business object	*BOprefix_BasicIDocType_ IDocSegmentName*	*BOprefix_BasicIDocType_ IDocSegmentName*
Data Record attribute	*IDocFieldName* or IDoc Field Description	When generating the BOs, the user has the choice to either choose IDoc segment field names or field descriptions as the BO attribute names.

Component names in the WebSphere business integration system support only alphanumeric characters and the underscore character (_). Therefore, when naming components in a generated business object definition, SAPODA replaces special characters in the IDoc segment field descriptions or field names with underscore characters. For example, SAPODA changes the spaces, parentheses, and periods in the following SAP description to underscores in the corresponding attribute name:

```
Partner function (e.g. sold-to party, ship-to party)
```

SAPODA represents the above description in the generated business object definition as:

```
Partner_function__e_g__sold_to_party__ship_to_party__
```

SAPODA guarantees that all attribute names in the business object definition are unique. If an IDoc has multiple fields with the same field descriptions, then SAPODA adds a counter suffix to the generated attribute name.

When naming an attribute, SAPODA prepends a string to the attribute name when the changed attribute name:

- Begins with a digit—prepends A_
- Begins with the underscore character (_)—prepends A

Important: You can modify attribute names at any time after you generate the business object. However, when you modify an attribute name, do not modify its application-specific information. The connector uses this text to identify the IDoc field to which the business object attribute corresponds. For more information, see "Application-specific information: Data record business object" on page 153.

Naming conventions for IDoc extensions

When SAPODA generates a business object definition based on an IDoc extension, it uses slightly different naming conventions than those described in "Business object naming conventions" on page 147. In this case, it includes the extension name as described in Table 26.

Table 26. Naming Conventions for Idoc extensions

IBM WebSphere business object or attribute	Name	Type
Parent wrapper business object	*BOprefix_BasicIDocType_ExtensionName*	n/a
Control Record business object	`Control_record`	`sap_idoccontrol`
Data Record business object	`Data_record`	*BOprefix_BasicIDocType*`_cwdata`
Data Record child business object	*BOprefix_BasicIDocType_ ExtensionName_IDocSegmentName*	*BOprefix_BasicIDocType_ ExtensionName_IDocSegmentName*
Data Record attribute	*IDocFieldText* or *IDocFieldName*	String

For the syntax of AppSpecificInfo property that specifies the extension, see "Parent wrapper business object."

Important: When InterChange System is the integration broker, be careful when you load a business object definition for an IDoc extension into the repository. You might encounter conflicts if a business object definition for the basic IDoc Type already exists in the repository and its name matches the basic IDoc Type plus extension. You must manually resolve these conflicts.

Parent wrapper business object

The name of the parent wrapper business object is the basic IDoc type prefixed by a user-defined prefix followed by an underscore (_), for example `sap_`. The parent wrapper business object contains four attributes: `Dummy_key`, `Control_record`, `Data_record`, and `TransactionId`.

The `Control_record` and `Data_record` attributes represent single-cardinality child business objects.

The type of the `Control_record` attribute is `sap_idoccontrol`. This business object definition is provided with the ALE Module.

The type of the `Data_record` attribute is *BOprefix_BasicIDocType*`_cwdata`. This business object definition contains one or more child business objects, depending on the IDoc segment definition of a basic IDoc type from the SAP application.

The value in the `TransactionId` attribute determines whether the connector manages TIDs when processing service call requests. If you do not want TID management for request processing, do not set the value for the TransactionID attribute.

The application-specific information of the parent wrapper business object indicates:

- The type of IDoc to be created

- The IDoc extension—Set only if the business object is generated from a customization of a basic IDoc type. For more information on generating the IDoc definition file, see "Before using SAPODA" on page 291.
- ALE Communication Partner information—Set only if your data requires more than one Partner type, Partner number, or Partner function.

Syntax

The AppSpecificInfo property of the parent wrapper object has the following syntax:

```
BasicIDocType [,Ext=ExtensionName
[,Pn=PartnerNumberOfRecipient [,Pt=
PartnerTypeOfRecipient[,Pf=PartnerFunctionOfRecipient
]]
```

Explanation of syntax

BasicIDocType
: Specifies the basic IDoc type

`Ext`
: Specifies the extension type

`Pn`
: Specifies the Partner number of the recipient

`Pt`
: Specifies the Partner type of the recipient

`Pf`
: Specifies the Partner function of the recipient

Example

```
AppSpecificInfo = ALEREQ01,Pn=ALESYS2,Pt=LS,Pf=EL
```

Control record business object

The ALE Module uses a generic control record business object definition for all IDocs. It contains a superset of attributes that are present in the 3.x version (SAP structure `EDI_DC`) and the 4.x version (SAP structure `EDI_DC40`) of the control record. The control record business object definition is provided with the ALE Module, and must be loaded into the business object repository. Use Business Object Designer to load the business object into the repository.

Note: Alternatively, if the IBM WebShpere InterChange Server is the integration broker, you can use the `repos_copy` command.

Table 27 lists the simple attribute properties of the control record business object.

Table 27. Properties of simple attributes in the control record business object

Property name	Description
Name	The value of the `Name` property is the modified value of the `TEXT` field in the IDoc definition. SAPODA replaces special characters (such as periods, slashes, and spaces) with underscores so that the name contains only alphanumeric characters and the underscore character (_), as described in "Business object naming conventions" on page 147
Type	Specifies the type of data. SAPODA sets the value to `String`.
MaxLength	SAPODA derives the value of `MaxLength` from the `LENGTH` field in the IDoc definition.
IsKey	SAPODA sets this property to `true` on the first attribute of a business object.
IsForeignKey	SAPODA sets the value to `false`.

Table 27. Properties of simple attributes in the control record business object (continued)

Property name	Description
IsRequired	The `IsRequired` property specifies whether an attribute must contain a value. SAPODA sets this property to `true` only on the `Name_of_table_structure` attribute in the control record object.
AppSpecificInfo	SAPODA derives the value from the NAME field in the IDoc definition.
DefaultValue	Specifies the value to assign to this attribute if there is no run-time value. SAPODA does not set a value for this property.

Important: When an attribute's value is set to either `CxIgnore` or `CxBlank` in the control record business object, the connector sets the value to a blank space for the IDoc control record.

Data record business object

An IDoc definition file has information about the structure of the IDoc, the IDoc segment hierarchy, and the fields that make up the segments. SAPODA uses the IDoc as input to generate the data record business object and its child business objects. The number of children depends on the IDoc segment definition of the basic IDoc type from the SAP application.

The top level of the data record business object corresponds to the basic IDoc type. This top-level business object contains an attribute that represents a child business object or an array of child business objects (one for each IDoc segment). The structure and hierarchy of the child business objects match that of the IDoc segments in the basic IDoc type.

Generating an IDoc from the system using SAPODA creates the data record object and its child business objects by making calls into the SAP system itself. Fields from an IDoc definition file are used in this section to help illustrate how different properties of a business object are set. Generating an IDoc from the system uses corresponding fields from the calls made into the SAP system.

This section describes:

- "Attributes: Data record business object"
- "Application-specific information: Data record business object" on page 153
- "Illustration of the relationship between business object and IDoc" on page 154

Attributes: Data record business object

Table 28 describes the properties of each simple attribute in the data record business object. SAPODA generates the properties described below.

Table 28. Simple attributes: data record business object

Property name	Description
Name	The value of the `Name` property is the modified value of the `NAME` or `TEXT` field in the IDoc definition. SAPODA replaces special characters (such as periods, slashes, and spaces) with underscores so that the name contains only alphanumeric characters and the underscore character (_), as described in "Business object naming conventions" on page 147.
Type	Specifies the type of data. SAPODA sets the value to `String`.

Table 28. Simple attributes: data record business object (continued)

Property name	Description
MaxLength	SAPODA derives the value of `MaxLength` from the `LENGTH` field in the IDoc definition.
IsKey	SAPODA sets this property to `true` on the first attribute in each business object. For every other attribute, SAPODA sets the value to `false`.
IsForeignKey	SAPODA sets the value to `false`.
IsRequired	Specifies whether an attribute must contain a value. SAPODA sets the value to `false`.
AppSpecificInfo	SAPODA sets the value of the `AppSpecificInfo` property to the value of the `Name` field in the IDoc definition prepended by the offset value and the `+` character; for example, for a segment field named `SIGN` with an offset of `40`, it sets the following value for `AppSpecificInfo`: `40+SIGN`For more information, see "Application-specific information: Data record business object" on page 153.
DefaultValue	Specifies the value to assign to this attribute if there is no run-time value. SAPODA does not set a value for this property.

Important: Simple attributes in the data record business object can have two special values: `CxIgnore` and `CxBlank`. Simple attributes set to `CxIgnore` or `CxBlank` are represented by blank spaces in the segment data string. SAP processes these attributes by placing one space character in the application field.

Table 29 describes the properties of each attribute in the data record business object that represents a child or array of child business objects. SAPODA generates the properties described below.

Table 29. Attributes that represent child business objects

Property name	Description
Name	SAPODA sets the value to `BOprefix_BasicIDocTypeIdocSegmentName`; for example, `SAP_E2ALER1001`
Type	SAPODA sets the value to: `BOprefix_BasicIDocTypeIdocSegmentName`
ContainedObjectVersion	SAPODA sets the value to `1.0.0`.
Relationship	SAPODA sets the value to `containment`.
IsKey	SAPODA sets the value to `false`.
IsForeignKey	SAPODA sets the value to `false`.
IsRequired	The `IsRequired` property specifies whether a child business object must exist. SAPODA sets the value to `false` if the value of the STATUS field for the corresponding segment in the IDoc definition has a value of `OPTIONAL`. SAPODA sets this property to `true` if the STATUS field in the IDoc definition has a value of `MANDATORY`.
AppSpecificInfo	The AppSpecificInfo property contains information on the hierarchy level and minimum and maximum number of allowed occurrences of a segment. For more information, see "Application-specific information in attributes that represent children" on page 153.
Cardinality	If the value of the `LOOPMAX` field in the IDoc definition is `1`, SAPODA sets the value to `1`. If the value of `LOOPMAX` is greater than `1`, SAPODA sets the value to `n`.

Application-specific information: Data record business object

This section describes how connector uses the value of the AppSpecificInfo property:

- "Application-specific information at the business-object Level"
- "Application-specific information in simple attributes"
- "Application-specific information in attributes that represent children"

Application-specific information at the business-object Level: The connector uses the value of the AppSpecificInfo property at the business-object level of the data record and each of its children to obtain the name of the associated Idoc and its segments:

- The syntax of the application-specific information on the data record business object is:

  *IDocType*_CWDATA

  For example, given an IDoc named ALERQ01, SAPODA creates the value of the AppSpecificInfo property as ALERQ01_CWDATA.
- The value of the application-specific information on the children of the data record business object is the corresponding segment name. For example, given IDoc ALERQ01 with two segments named E2ALER1001 and E2ALEQ1, SAPODA automatically creates the value of the AppSpecificInfo property for the two child business objects as:
  - First child: E2ALER1001
  - Second child: E2ALEQ1

Application-specific information in simple attributes: The connector uses the value of the AppSpecificInfo property of simple attributes to obtain the field name in SAP and its position (offset) in the data string.

The offset value is the position of the first character of the attribute value in the data string. The offset value is calculated by subtracting the value in the BYTE_FIRST value of the first field in the IDoc definition from the BYTE_FIRST value of the given attribute. This value is used with the MaxLength property to build the data string for the IDoc segment.

The syntax of the AppSpecificInfo property of simple attributes is:

OffsetNumber+IDoc*FieldName*

For example, a segment field named SIGN with an offset of 40 has the following value for AppSpecificInfo:

```
40+SIGN
```

Application-specific information in attributes that represent children: The connector uses the value of the AppSpecificInfo property of attributes that represent a child or array of child business objects to obtain information on the hierarchy level and minimum and maximum number of allowed occurrences of a segment. SAPODA sets the AppSpecificInfo property for these attributes by obtaining information from the LEVEL, LOOPMIN and LOOPMAX fields in the IDoc definition.

Illustration of the relationship between business object and IDoc

Figure 19 illustrates the relationship between the WebSphere data record business object and the IDoc definition from an SAP application.

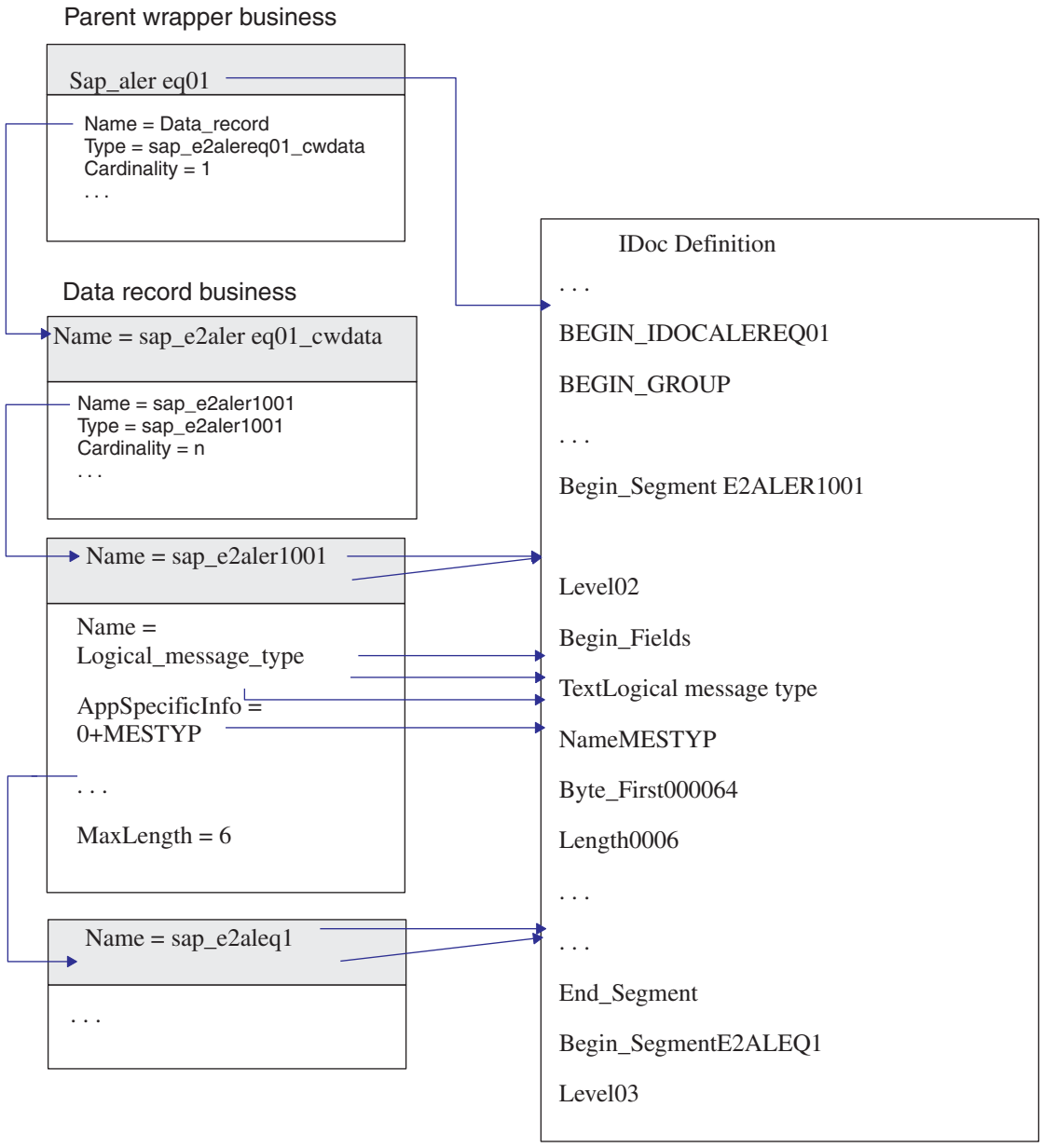


Figure 19. Relationship Between data record business object and IDoc Definition fields

Supported verbs

Verb support for the ALE Module is limited by the verbs that SAP supports through its ALE interface. SAPODA generates the Create, Update, Delete, and Retrieve verbs in the business object definition. Implementation of each verb requires knowledge of the ALE configuration within SAP.

SAPODA generates the `AppSpecificInfo` for the verbs and the `AleOutboundVerbs` meta-verb on the parent wrapper business object. However, it populates only one of the parameters of the `AppSpecificInfo` with values: it specifies the business

object handler to use for service-call request processing. For all other processing, you must manually modify the business object definition to add or remove specific information:

- When using the business object for event processing, you must specify values for the following the `AppSpecificInfo` properties:
  - Parent wrapper business object's verb—specify a value for those parameters that uniquely identify the verb. Depending on the requirements of your ALE configuration, specify the message type, message code, and message function. Make these changes after you import the business object definition into your repository.

    Important: SAPODA inserts the `AppSpecificInfo` value that specifies the business object handler, which the connector uses only for request processing. SAPODA does not insert values for the message parameters. If you are using the ALE Module for event processing, you must manually add the values for the message parameters.
  - Parent wrapper business object's `AleOutboundVerbs` meta-verb—a comma-separated list of verbs supported for event processing.
- When using the business object for request processing, you must specify a value for the following the `AppSpecificInfo` properties:
  - Parent wrapper business object's verb—specify the package and classname of the business object handler so that the connector can determine the appropriate business object handler. SAPODA inserts the following value into the `AppSpecificInfo` property of each standard verb: `AppSpecificInfo = sap.sapalemodule.VSapALEBOHandler`.
  - When using a wrapper business object to process multiple IDoc parent business objects, you must add the package and classname of the business object handler to the `AppSpecificInfo` property of each verb in the multiple IDoc wrapper business object.

  For each parent wrapper business object, SAPODA generates the Create, Retrieve, Update, and Delete verbs. For each of these verbs, it generates the following `AppSpecificInfo` values:

  ```
  sap.sapalemodule.VSapALEBOHandler,MsgType=;MsgCode=;MsgFunction=
  ```

Supporting multiple message types

To support multiple message types that map to the same basic IDoc type:

1. Generate as many business objects from the same IDoc as the number of message types you want to support. Note that the same business object needs to be saved with different names.
2. Configure the verb apptext to set `"MsgType= xxxxx"` to the proper message type in each object. The business object level ASI will be set to the same IDoc type in all of these objects.
3. Perform the necessary configurations in SAP. Maintain transaction WE82 to map message type to basic type.

Using the previous configuration, a single instance of the adapter suffices to support multiple mappings of the same basic IDoc type to various message types.

Two instances of the adapter cannot use the same set of MQ Queues for event processing. Create a set of MQ Queues for each adapter.

AppSpecificInfo property: Parent wrapper verb

The syntax of the AppSpecificInfo property of the parent wrapper business object's verb differs depending on whether the business object represents an application event or a service call request:

Application event syntax

```
[BOHandler],MsgType=messageType;MsgCode=[messageCode];MsgFunction=[messageFunction]
```

Note: The connector matches the values in the control record with the values specified in the verb's AppSpecificInfo property to determine the verb.

Service call request syntax

```
BOHandler[,MsgType=messageType;MsgCode=[messageCode];MsgFunction=[messageFunction]]
```

Explanation of syntax

BOHandler	Specifies the request-processing business object handler; the value defaults to the following: sap.sapalemodule.VSapALEBOHandler
MsgType	Specifies the message type configured for the IDoc in ALE
MsgCode	Specifies the message code configured for the IDoc in ALE; the connector requires a value only if MsgType does not uniquely identify the verb; however, specify a value if required by your ALE configuration
MsgFunction	Specifies the message function configured for the IDoc in ALE; the connector requires a value only if MsgType and MsgCode do not uniquely identify the verb; however, specify a value if required by your ALE configuration

AppSpecificInfo property: Parent wrapper meta-verb

In the AppSpecificInfo property of the parent wrapper business object's AleOutboundVerbs verb, specify those verbs the connector should support for application-event processing, separating verbs with a comma.

Important: SAPODA generates values for the Create, Retrieve, Update, and Delete verbs. After the definition has been generated, you must manually delete those verbs that you do not want the connector to support.

The following example instructs the connector to support the Create and Update verbs for processing application events:

```
[Verb]
Name = AleOutboundVerbs
AppSpecificInfo = Create, Update
[End]
```

Processing multiple IDocs with a wrapper business object

Note: This section is applicable only to service-call request processing.

When processing multiple IDocs, the ALE Module requires a wrapper business object as the top-level business object. The multiple IDoc wrapper business object contains an attribute that represents an array of IDoc parent wrapper business objects.

For each parent wrapper business object, SAPODA generates the Create, Retrieve, Update, and Delete verbs. For each of these verbs, it generates the following `AppSpecificInfo` values:

```
sap.sapalemodule.VSapALEBOHandler,MsgType=;MsgCode=;MsgFunction=
```

Figure 20 illustrates the relationship between a top-level wrapper object and it's child IDoc business objects.

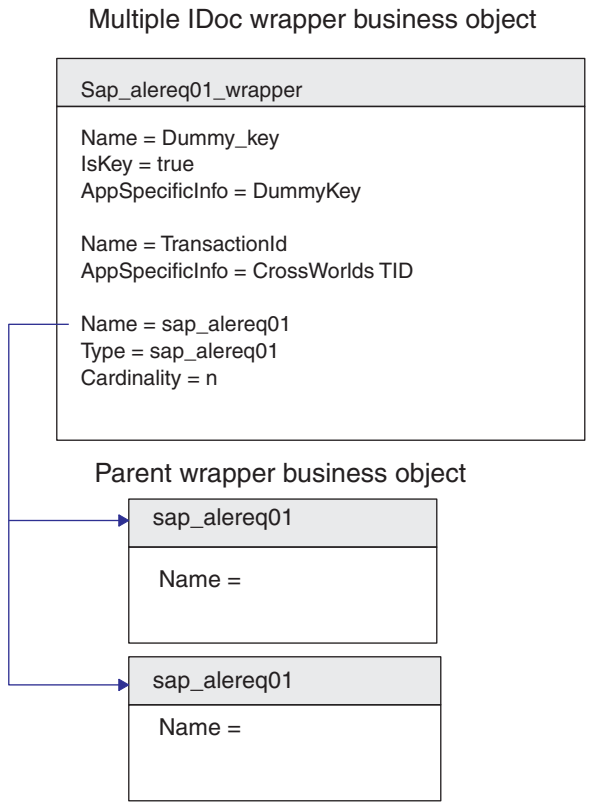


Figure 20. Wrapper business object containing child business objects

Multiple IDoc wrapper object example

The following is a sample definition of a multiple IDoc wrapper business object:

```
[BusinessObjectDefinition]
Name = sap_alereq01_wrapper
Version = 1.0.0
AppSpecificInfo =

[Attribute]
Name = Dummy_key
Type = String
Cardinality = 1
MaxLength = 1
IsKey = true
IsForeignKey = false
IsRequired = true
AppSpecificInfo = DummyKey
DefaultValue =
[End]

[Attribute]
Name = TransactionId
Type = String
```

```
Cardinality = 1
MaxLength = 1
IsKey = false
IsForeignKey = false
IsRequired = false
AppSpecificInfo = CrossWorlds TID
DefaultValue =
[End]

[Attribute]
Name = sap_alereq01
Type = sap_alereq01
ContainedObjectVersion = 1.0.0
Relationship = Containment
Cardinality = n
MaxLength = 255
IsKey = false
IsForeignKey = false
IsRequired = false
AppSpecificInfo =
DefaultValue =
[End]

[Verb]
Name = Create
AppSpecificInfo = sap.sapalemodule.VSapALEBOHandler,MsgType=;MsgCode=;MsgFunction=
[End]

[Verb]
Name = Retrieve
AppSpecificInfo = sap.sapalemodule.VSapALEBOHandler,MsgType=;MsgCode=;MsgFunction=
[End]

[Verb]
Name = Update
AppSpecificInfo = sap.sapalemodule.VSapALEBOHandler,MsgType=;MsgCode=;MsgFunction=
[End]

[Verb]
Name = Delete
AppSpecificInfo = sap.sapalemodule.VSapALEBOHandler,MsgType=;MsgCode=;MsgFunction=
[End]
```

Multiple IDoc wrapper: Attribute that represents the child business object

Table 30 lists and describes the properties of the attribute that represents the child business object in the multiple IDoc wrapper business object.

Table 30. Multiple IDoc wrapper: attribute that represents child business object

Property name	Description
Name	Set the value to the name of the parent business object generated by SAPODA.
Type	Set the value to the name of the parent business object generated by SAPODA.
ContainedObjectVersion	Set the value to `1.0.0`.
Relationship	A child business object is contained by a parent business object; therefore, the value is `containment`.
IsKey	Set the value to `false`.
IsForeignKey	Set the value to `false`.
IsRequired	Set the value to `false`.
AppSpecificInfo	This property is not used for the attribute that represents child business objects in the ALE Module.

Table 30. Multiple IDoc wrapper: attribute that represents child business object (continued)

Property name	Description
Cardinality	Set the value of the attribute in the top-level wrapper business object that represents the IDoc parent business object to cardinality `n`.

Part 4. BAPI module

Chapter 14. Overview of the BAPI Module

This chapter introduces the SAP BAPI ModuleIBM WebSphere Business Integration Adapter for mySAP.com. The BAPI Module enables an integration broker to send business objects to SAP R/3 application versions 3.x using BAPIs.

BAPIs are SAP's standardized Business Application Programming Interfaces that enable third parties to interact with SAP R/3 applications. They are implemented as RFC-enabled function modules for an SAP business object's methods.

This chapter contains the following sections:

Note: A BAPI is an RFC-enabled function in an SAP application. In addition to BAPIs, the BAPI Module can be used to support any RFC-enabled function.

BAPI Module components

The BAPI Module is a connector module written in Java that supports native BAPI calls directly to an SAP R/3 application. It extends the vision connector framework by implementing the VisionConnectorAgent and VisionBOHandler classes. The BAPI Module uses the SAP RFC libraries written in Java and C, which enables external programs to communicate with an SAP R/3 application.

Figure 21 illustrates the overall architecture of the BAPI Module. The BAPI Module is made up of the connector framework, the connector's application-specific component for BAPI, and BAPI-specific business object handler connector modules, as well as the SAP RFC Library.

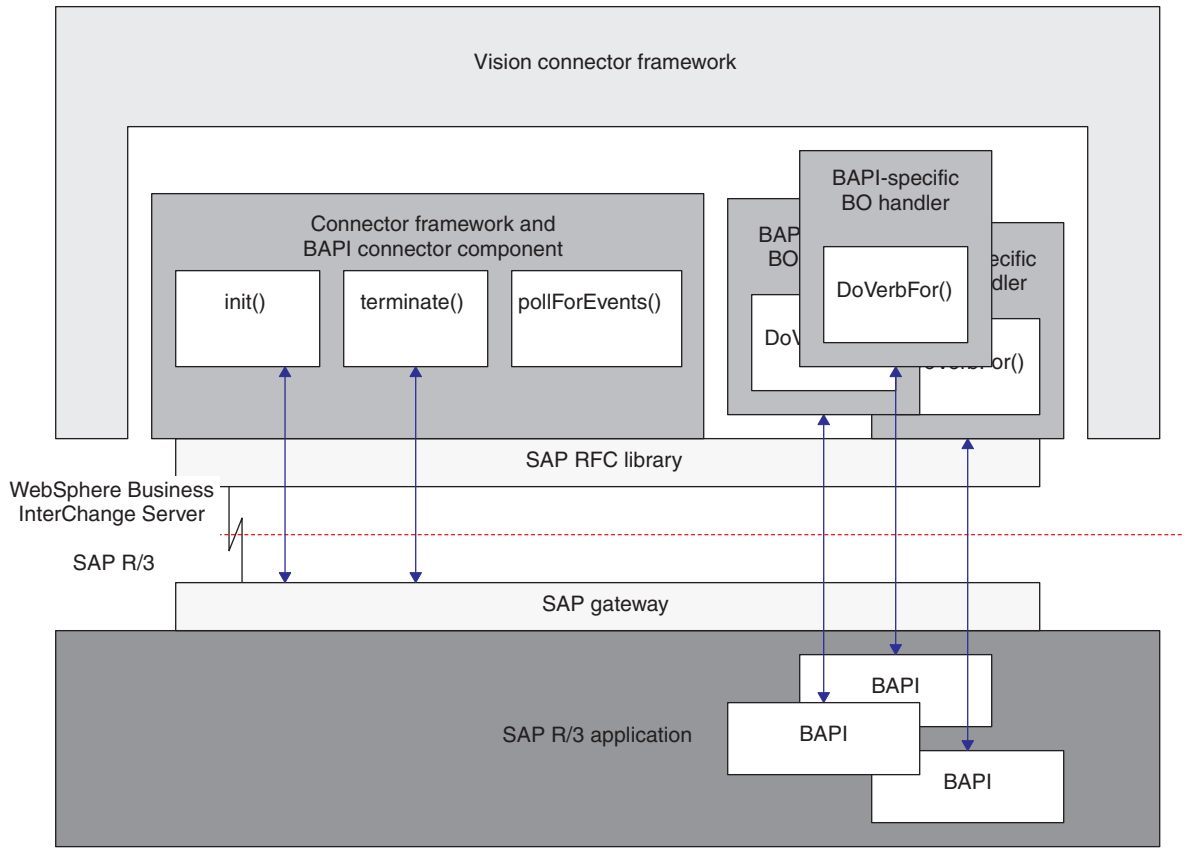


Figure 21. BAPI Module architecture

The BAPI Module components:

- Open an RFC connection to the SAP R/3 application using the SAP RFC library and the SAP Gateway.
- Handle requests from the integration broker and call BAPIs in an SAP R/3 application.
- Terminate connections to the SAP R/3 application.

How the BAPI Module works

The BAPI Module implements the `init()`, `terminate()`, `pollForEvents()`, and `doVerbFor()` methods. However, the `pollForEvents()` method is not used because the BAPI Module supports request operations only.

Initialization and Termination

The `init()` method opens an RFC connection with the SAP R/3 application through the SAP Gateway. If the connector fails to initialize, it terminates using the `terminate()` method. The connector terminates by disconnecting the connection to the SAP Gateway.

Business object processing

A single implementation of the `doVerbFor()` method in the vision connector framework's business object handler initiates all business object requests. The vision business object handler processes all of the business objects passed between the BAPI Module and the integration broker. In the BAPI Module, a BAPI-specific

business object handler supports only one BAPI; therefore, for each supported BAPI in the SAP R/3 application, you must have an associated BAPI-specific business object handler.

The vision business object handler uses the verb application-specific information of a business object to invoke the appropriate BAPI-specific business object handler. The BAPI parameter names and formats are hard-coded in the BAPI-specific business object handler so that the business object handler can make an RFC call to the appropriate BAPI.

Figure 22 illustrates business object processing for the BAPI Module.

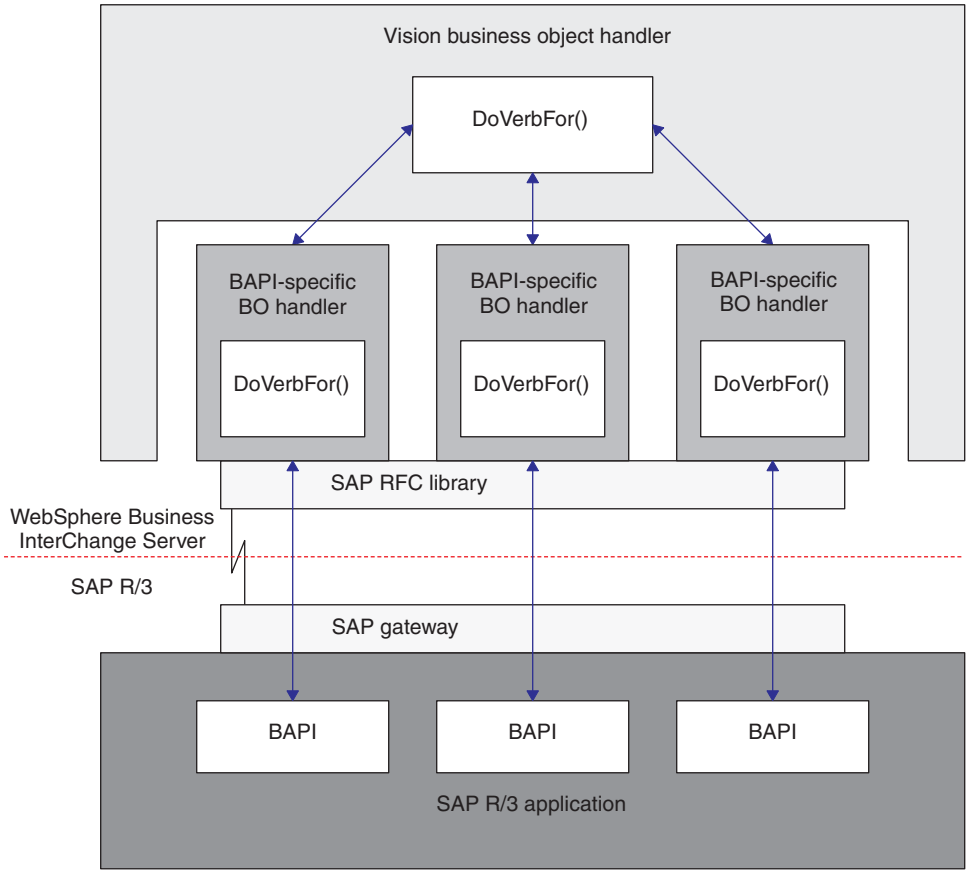


Figure 22. Business object processing for the BAPI Module

Once invoked by the vision business object handler, the BAPI-specific business object handler executes in the following manner:

1. Receives the WebSphere business object for SAP from the vision business object handler.
2. Populates the BAPI parameters with business object data.
3. Executes a BAPI call using RFC and passes the BAPI parameters to the SAP R/3 application. The business object handler waits for the business object data to be returned.
4. Receives the business object data (BAPI parameters).
5. Converts the BAPI parameters back to WebSphere business object data.
6. Passes the business object to the Vision business object handler and ultimately to the integration broker.

Note: If a BAPI Module has a Return Structure or Return Table, the connector checks for the message types `A` (abort) and `E` (error) to determine if the service call request processed successfully. A message type `A` or `E` indicates that the service call request failed to process. If a BAPI does not have a Return Structure or Return Table, you must implement your own error handling. The error message or messages, within the structure or table, are returned in the return status descriptor.

Supporting BAPIs

The business object generation utility, SAPODA, generates business object definitions that support BAPIs. SAPODA interprets the interface of a BAPI, maps its parameters to the business object attributes, and adds the application-specific information for each attribute.

Also, for each WebSphere business object definition, you must generate an associated BAPI-specific business object handler using SAPODA. For more information on Developing business objects and BAPI-specific business object handlers, see Chapter 16, “Developing business objects for the BAPI Module,” on page 169.

Note: Some BAPIs do not have single field parameters that correspond to simple attributes in the WebSphere business object. The connector requires every top-level business object to have a simple attribute that serves as the key attribute. Therefore, when generating a business object and business object handler from a BAPI without a single field parameter, SAPODA creates a key attribute named Dummy_key in the top-level business object, marks it as the key attribute, and adds `dummy_key` as the application-specific information of this attribute. Dummy_key provides the connector with a key attribute so that it can process the business object. However, the connector ignores the value of the Dummy_key attribute when modifying application data.

Chapter 15. Configuring the BAPI Module

This chapter describes the configuration of the BAPI Module and assumes that all of the necessary files were installed when the IBM WebSphere Business Integration Adapter for mySAP.com was installed. For more information on installing the connector, see Chapter 2, “Installing and configuring the connector,” on page 11..

This chapter contains the following sections:

BAPI Module directories and files

The BAPI Module directory and files are contained in the `\connectors\SAP\` directory. Table 31 lists the directory and file used by the BAPI Module.

Table 31. BAPI Module directory and file

Directory/filename	Description
`\bapi\client`	Directory containing the runtime files for the connector. All BAPI-specific BO Handler class files must be copied into this directory.
`CWSAP.jar`	Connector class file

BAPI Module configuration properties

You must configure the BAPI Module before it can start operating. To configure the BAPI Module, set the standard and connector-specific connector configuration properties. For more information on configuring the connector configuration properties, see “Configuring the connector” on page 17 and Appendix B, “Standard configuration properties for connectors,” on page 241..

Chapter 16. Developing business objects for the BAPI Module

This chapter describes business objects required for the BAPI Module. It also discusses how the business object generation utility, SAPODA, generates the definitions. The chapter assumes you understand how the connector processes business objects. For more information on business object processing in the BAPI Module, see Chapter 14, "Overview of the BAPI Module," on page 163..

This chapter contains the following sections:

Note: This chapter describes business objects that support BAPIs; however, the BAPI Module can be used to support any RFC-enabled function.

Background information

Business object development for the BAPI Module requires creation of the following for each supported BAPI:

- An application-specific business object
- An associated BAPI-specific business object handler

SAPODA facilitates the process of developing business objects and BAPI-specific business object handlers. SAPODA uses the SAP application's native definitions as a template when generating business object definitions for the integration brokerIBM WebSphere Business Integration Adapter for mySAP.com.

Important: SAPODA must have access to the BAPI in an SAP R/3 system to retrieve the BAPI interface.

Note: SAP supports many methods that can be mapped to the standard verbs (Create, Update, Delete, and Retrieve) that the connector supports. You can develop business objects and BAPI-specific business object handlers to support any method used by BAPIs.

Business object naming conventions

A BAPI interface consists of importing, exporting, and table parameters, where:

- Importing parameters are passed to the BAPI.
- Exporting parameters are returned from the BAPI.
- Table parameters are passed in either direction.

Some BAPIs may not have all of the types of parameters. For example, a BAPI may have importing and table parameters only.

SAPODA automatically maps the BAPI importing, exporting, and table parameters to attributes in WebSphere business objects for SAP as described in Table 32.

Table 32. Naming conventions: WebSphere business objects for SAP

Business object	BAPI interface
Top-level business object	*BOprefix_BAPIname* **Note:** The illustrations in this chapter use `SAP_` or `sap_` as the business object prefix. You can specify your own meaningful prefix when you create your business object definitions.
Attribute	*FieldDescription*
Child business object	*BOprefix_BAPIParameterName*

SAPODA guarantees that all attribute names in the business object definition are unique. If a BAPI has multiple parameters with the same field description, SAPODA adds a counter as the suffix to the generated attribute name.

When naming an attribute from a BAPI parameter, SAPODA prepends a string to the attribute name when the changed attribute name:

- Begins with a digit—prepends `A_`
- Begins with the underscore character (_)—prepends `A`

Important: You can modify the attribute names at any time after you generate the business object definition. However, when you modify an attribute name, do not modify the application-specific information. The connector uses this information to identify the BAPI parameter to which the attribute corresponds. For more information on the application-specific information, see "AppSpecificInfo for attributes" on page 174.

Business object structure

The connector uses a BAPI-specific business object handler to map each business object attribute to a BAPI parameter. The connector, each business object, and each BAPI-specific business object handler are metadata-driven. The application-specific information provided in the metadata of each business object and business object handler allows you to add connector support for a new business object and its handler without modifying connector code. Instead:

- The connector uses the verb application-specific information of the top-level business object to instantiate the appropriate BAPI-specific business object handler
- The business object handler uses the attribute application-specific information of each business object to map between each attribute and its parameter

Each BAPI-specific business object handler supports both single- and multiple-cardinality relationships between business objects.

A business object based on a BAPI can contain no more than two levels of hierarchy. Therefore, all BAPI simple parameters correspond to attributes of the top-level business object, and BAPI structure and table parameters correspond to child business objects.

Table 33. Correspondence between BAPIs and WebSphere business objects for SAP

BAPI interface parameter	WebSphere business object for SAP
Simple field	Attribute of the top-level business object
Structure	Single-cardinality child business object
Table	Multiple-cardinality child business objects

Note: Importing and exporting parameters can be simple field or structure parameters.

Figure 23 illustrates the association between a business object and a BAPI. The figure illustrates a fragment of the sap_bapi_salesorder_createfromdat2 business object, which corresponds to the `BAPI_SALESORDER_CREATEFROMDAT2` BAPI.

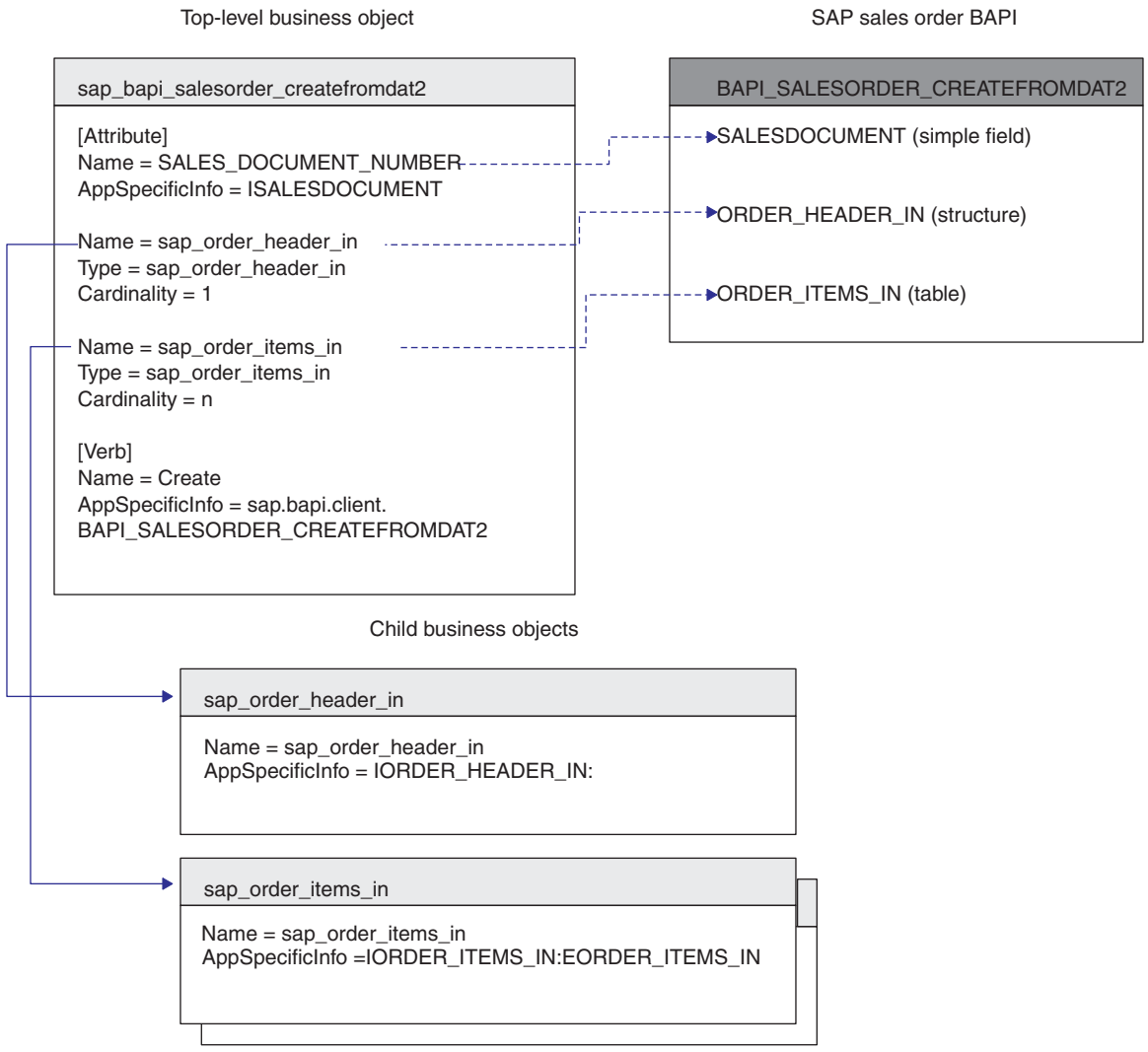


Figure 23. Mapping between a business object and a BAPI

Supported verbs

The BAPI Module supports the standard verbs (Create, Update, Delete, and Retrieve) used by the WebSphere business integration system. For each supported verb, a BAPI can have an associated method. Most BAPIs support one of the following operations: create, retrieve, update, and delete.

Business object attribute properties

The properties of the attributes of a top-level business object differ depending on whether the attribute represents a simple value, or a child or an array of child business objects.

- Table 34 lists and describes the properties of simple attributes of a top-level business object.
- Table 35 lists and describes the attributes that represent a child or array of child business objects.

SAPODA generates the attribute properties as described in each table.

Table 34. Simple attributes properties: Top-level business object

Property name	Description
`Name`	Derived from the description or name of the BAPI parameter. SAPODA replaces special characters (such as periods, slashes, and spaces) with underscores.
`Type`	Specifies the type of data. SAPODA sets the value to `String`.
`MaxLength`	Specifies the field length of the BAPI parameter.
`IsKey`	Specifies whether the attribute is the key. The first simple attribute of a business object defaults to the key attribute. The connector does not support using an attribute that represents a child business object or an array of a child business objects as a key attribute. Therefore, if the BAPI provides only structure and table parameters, you must insert a simple attribute as the first attribute. SAPODA inserts the `Dummy_key` attribute as the first attribute, marks it as the key attribute, and sets appropriate values. Do not modify those values.
`IsForeignKey`	SAPODA sets the value to `false`.
`IsRequired`	Specifies whether an attribute must contain a value. SAPODA sets the value to `false`.
`AppSpecificInfo`	Contains the name of the BAPI parameter that corresponds to the associated attribute. The format is: `I`*`ABAPFieldName`*`:E`*`ABAPFieldName`* For more information on the application-specific information, see “Business object application-specific information” on page 174.
`DefaultValue`	Specifies the value to assign to this attribute if there is no run-time value. SAPODA does not set a value for this property.

Table 35 lists and describes the attributes that represent a child or array of child business objects. SAPODA generates the properties described below.

Table 35. Properties of an attribute that represents a child or children

Property name	Description
`Name`	The value is the name of the structure or table parameter. The format is: `BOprefix_BAPIParameterName`.

Table 35. Properties of an attribute that represents a child or children (continued)

Property name	Description
Type	The value is the type of child business object; in other words, the type is BOprefix_BAPIParameterName.
ContainedObjectVersion	SAPODA sets the value to 1.0.0.
Relationship	SAPODA sets the value to containment.
IsKey	SAPODA sets the value to false.
IsForeignKey	SAPODA sets the value to false.
IsRequired	Specifies whether an attribute must contain a value. SAPODA sets the value to false.
AppSpecificInfo	Contains the name of the BAPI parameter that corresponds to the associated attribute. The format is: I*BAPIParameterName*:E*BAPIParameterName* For more information on the application-specific information, see "AppSpecificInfo for attributes" on page 174.
Cardinality	BAPI structure parameters have single cardinality (1) and BAPI table parameters have multiple cardinality (n).

Important: Simple attributes can have two special values: CxIgnore and CxBlank. When a business object is sent to the BAPI Module as a service call request and the business object has simple attributes set to CxIgnore or CxBlank, it is as if those attributes are invisible to the BAPI Module. However, the SAP application initializes such an attribute to its ABAP data type. The BAPI Module converts all returned blank values to CxIgnore.

Initializing attribute values

Every field in SAP has an initial value. When the connector receives a service call request, the BAPI-specific business object handler populates most of the BAPI interface parameters with the values listed in Table 36. The one exception is the character data type. The business object handler converts a CxIgnore in the business object attribute to a space in the SAP field. If you want any other value to be converted to CxIgnore, the component that creates the business object must perform the conversion. For example, when the WebSphere Inter Change Server is the integration broker, modify the map to handle this conversion.

Table 36 provides initial values set by the business object handler.

Table 36. Initial field values in SAP

Data type	Description	Initial value set by business object handler
C	Character	space
N	Numeric string	*000...*
D	Date (YYYMMDD)	00000000
T	Time (HHMMSS)	000000
X	Byte (hexadecimal)	X00
I	Integer	0
P	Packed number	0
F	Floating point number	0.0

Business object application-specific information

Application-specific information in business object definitions provides the BAPI Module with application-dependent instructions on how to process business objects. These instructions are specified at the business-object level, at the attribute level (both for simple attributes and for attributes that represent a child or array of child business objects), and for verbs.

AppSpecificInfo for the verb of the top-level business object

The connector uses the value of the verb application-specific information in the top-level business object to call the appropriate BAPI-specific business object handler. The value of the `AppSpecificInfo` property specifies the package and classname for the BAPI-specific business object handler. The format is as follows:

```
AppSpecificInfo = bapi.client.BOHandler
```

where `BOHandler` is the name of the class. By default, SAPODA uses the name of the BAPI as the name of the class. SAPODA automatically adds the application-specific information to the top-level business object.

Important: You must include the value `client` before the business object handler name to identify that the BAPI-specific business object handler acts as a client.

For example, if you are supporting the `SALES_ORDER_CREATEFROMDAT2` BAPI, then the application-specific information is as follows:

```
AppSpecificInfo = bapi.client.sales_order_createfrom dat2
```

AppSpecificInfo for attributes

The connector uses the value of an attribute's application-specific information to determine which importing, exporting, and table parameters to use. The value of this property contains the prefix `I` (for importing parameters) or `E` (for importing parameters). The prefix indicates whether the attribute value is used to pass data into or out from the SAP application.

Because structure parameters can be either importing or exporting, they use either an `I` or an `E` before the parameter value. Because table parameters can pass data to and return data from a BAPI, they can have both `I` and `E` parameter values.

Important: Always use a colon (`:`) separator when you specify parameter values with `I` and `E`. If specifying only an importing value, the colon must follow the value. If specifying only an exporting value, the colon must precede the value. If specifying both values, the colon follows the importing value and precedes the exporting value.

Table 33 illustrates the correspondence between a business object and an example BAPI named `BAPI_EXAMPLE`. In the example, the simple attributes (Attribute_1, Attribute_2, and Attribute_3) specify only an importing or exporting parameter. The attribute that represents a child business object (Child_1) corresponds to an exporting structure parameter. The attribute that represents an array of child business objects (Child_2) corresponds to a table parameter.

Each child business object has a simple attribute that corresponds to a field of the corresponding structure or table (Attribute_11 and Attribute_14, respectively). You can find these fields by looking at the details of the BAPI.

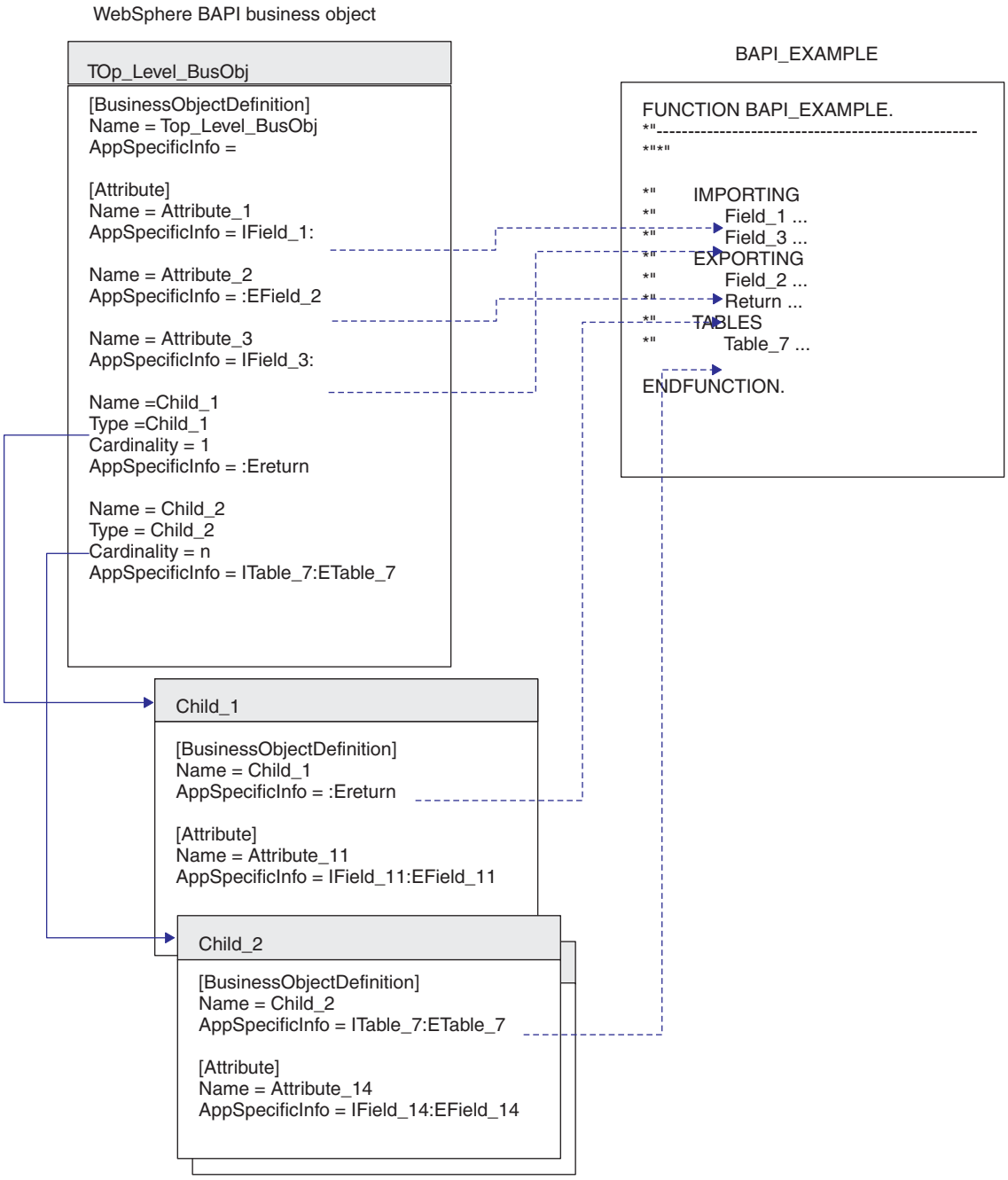


Figure 24. Correspondence between a business object and an example BAPI

Table 37 identifies the format of the application-specific information for specific kinds of attributes.

Table 37. AppSpecificInfo format for specific kinds of attributes

AppSpecificInfo Format	Attribute Type
I*ParameterName*:E*ParameterName*	Simple
I*TableName*:E*TableName*	Represents a child business object mapped to a table parameter
I*StructureName*:E*StructureName*	Represents a child business object mapped to a structure parameter

Table 37. AppSpecificInfo format for specific kinds of attributes (continued)

AppSpecificInfo Format	Attribute Type
I*FieldName*:E*FieldName*	Represents an attribute of a child business object mapped to a field in a table or structure parameter

SAPODA automatically generates the appropriate application-specific information for the business object definition. It is recommended that you do not change the parameter names of the generated application-specific information.

Using generated business object definitions and business object handlers

Use SAPODA to generate business object definitions and business object handlers for each RFC-enabled function that you want to support. You can use the generated objects without any modifications. However, you can manually edit these objects to refine the functionality.

After the objects are generated, you must add the business object definition and its corresponding BAPI-specific business object handler to your WebSphere business integration system's runtime environment.

- Use Business Object Designer to copy the business object definition into your repository.

  Note: Alternatively, if the WebSphere InterChange Server is the integration broker, you can use the `repos_copy` command to load the definition into the repository.
- Use a system command to copy the BAPI-specific business object handler files to the following directory under the product directory:

  `\connectors\SAP\bapi\client`

The BAPI-specific business object handler files are:

- *BAPI Name*`.java`
- *BAPI Name*`.class`

For example, given the `BAPI_SALESORDER_CREATEFROMDAT2` BAPI and a user-specified prefix of `sap_`, SAPODA generates the following:

- `sap_bapi_salesorder_createfromdat2` (business object definition that includes all child business objects)
- `Bapi_salesorder_createfromdat2.java`
- `Bapi_salesorder_createfromdat2.class`

Important: You can modify the name of the generated business object as well as the name of its child business objects. To do so, you must edit the definition as a text file rather than in Business Object Designer. If you do change a business object's name, ensure that you also modify all references to the names that you change. Also, if you modify the names of the generated `.class` file for the business object handler, you must maintain the changes for the application-specific information for the associated business object.

Note: For BAPIs and RFC-enabled ABAP functions that are developed in a development namespace, `SAPODA` removes or replaces `"/"` characters in the

function name with "_" when naming the business object definition, .java, and .class files. SAPODA removes the "/" character only when it is the first character of the name. Although the definition name or file name does not contain this character, the code still accurately calls the specified function with its proper name containing the "/" characters. Also, when a function name begins with a digit, SAPODA prepends the name with the string Rfm_.

Tips and tricks

This section describes the following tips and tricks for developing business objects and BAPI-specific business object handlers:

Multiple business objects contain the same return business object

Most BAPIs use the same name for the return object. When SAPODA generates a business object definition, it creates a child business object to represent this return object. If multiple business object definitions contain an identically named child business object, you can add that child business object into the repository only once, or copy only a single definition file into the repository directory.

To enable multiple business objects to contain the return business object, you must modify the name of the return business object to be unique for each business object.

To rename the return business object, modify the definition of each business object definition that contains it. The definition of the child business object is contained in the same definition file as its parent.

To rename the child, do the following:

1. Open the definition file for the top-level business object in a text editor.
2. Locate the definition of the BOprefix_return child business object.
3. Change the child's name to be unique. For example, append a number to the text (sap_return_2).
4. Change all references in the definition to refer to the newly named child. For example, change the value of the Type property for every attribute that represents the child business object.
5. Save the changed definition file.
6. Use Business Object Designer to load the newly named child business object into the repository.

   Note: Alternatively, if the WebSphere Integration Server is the integration broker, you can use the repos_copy command to load the definition into the repository.

Generated business object definition contains unnecessary attributes and child business objects

SAPODA interprets all BAPI interface parameters and, for each one, it creates a corresponding business object attribute or child business object. To increase performance of business object processing, remove from the business object definition all attributes and business objects that are not required.

Note: SAPODA facilitates graphically removing all optional attributes and child business objects before definition generation. For more information, see Chapter 16, "Developing business objects for the BAPI Module," on page 169.

To increase performance of business object processing, you can also remove from the application-specific information all importing and exporting table parameter values that are not required.

After definition generation, you can use Business Object Designer to manually edit the business object definition if you require other changes. However, be careful that you remove only attributes that you absolutely will not be using.

Generated business object names are too long or fail your naming conventions

SAPODA uses the name of the BAPI function module to generate the name of the business object definition. You can use a text editor to modify a business object's name.

Important: If you do change the name, ensure that you modify all references to the name as well. However, do not modify the parameter names of the generated application-specific information.

To change a generated business object's name:

1. Save the definition to a file.
2. Use a text editor to shorten or change the name.
3. Use Business Object Designer to load the newly named child business object into the repository.

   Note: Alternatively, if the WebSphere Integration Server is the integration broker, you can use the `repos_copy` command to load the definition into the repository.

Generated AppSpecificInfo for table parameters specify unnecessary parameters

Table parameters can be both importing and exporting parameters. If you do not require importing or exporting of values for a table parameter, you can remove it from the application-specific information.

For example, for a create operation, if you do not need to return the table data from the SAP application after the create operation has completed, you can remove the exporting parameter value (such as E*table name*).

For a retrieve operation, you do not need to specify any importing table parameters. Therefore, you can remove the importing parameter value (such as I*table name*).

Note: You must remove the unneeded value from the `AppSpecificInfo` of the attribute in the parent that represents the child as well as from the `AppSpecificInfo` at the business-object level of the child business object. Do not remove the colon (`:`).

For example, to remove the `ETable_7` exporting parameter in Figure 24 on page 175, you would do the following:

1. In the Child_2 attribute of the Top_Level_BusObj business object, change the attribute's AppSpecificInfo value to:

   ```
   ITable_7:
   ```

2. In the AppSpecificInfo at the business-object level of the Child_2 business object, change the value to:

   ```
   ITable_7:
   ```

3. In the AppSpecificInfo for each attribute of the child business object, using Attribute_14 as an example, change the value to:

   ```
   IField_14:
   ```

Using custom business object handlers

You can use custom business object handlers with the BAPI module. Reasons for choosing to write custom business object handlers include the following:

- Implementing custom error handling.
- Locking an object before modifying it. To achieve this you need to modify the generated business object handler as follows:
  1. Call ENQUEUE BAPI to lock the object.

     ```
     BAPI_EMPLOYEE_ENQUEUE
     ```

  2. Call the actual BAPI.

     ```
     BAPI_EMPLOYEE_UPDATE
     ```

  3. Call DEQUEUE BAPI to unlock the object.

     ```
     BAPI_EMPLOYEE_DEQUEUE
     ```

  Note: All calls should go into the same BAPI business object handler and should use the same JCO.Client for making the RFC call. For details on the specifics of coding RFC calls please refer to the generated business object handler.
- Calling multiple BAPIs using the same JCO.Client.

Note: There is no support for custom business object handlers.

Creating custom business object handlers

There are two ways to create custom business object handlers: modifying a generated business object handler or writing a business object handler from scratch.

Modifying a generated business object handler

SAP ODA by default generates business object handlers when you generate business objects. In addition to compiled class files, the ODA generates Java source files for the business object handlers. You can customize the generated Java source and create with your own business object handler.

Writing a business object handler

We do not recommend you to take this approach. If you still need to take this path please use the generated business object handler as a template because it provides calls to utility methods.

You can also use the following template to create a batch file for Windows platform to compile the custom business object handler:

```
REM @echo off
REM call "%CROSSWORLDS%"\bin\CWODAEnv.bat
setlocal
set WBIA="%CROSSWORLDS%"\lib\WBIA\4.2.0\WBIA.jar
set CWLIB="%CROSSWORLDS%"\lib\CrossWorlds.jar
set AGENT="%CROSSWORLDS%"\ODA\SAP\SAPODA.jar
set
JCO_JAR="%CROSSWORLDS%"\ODA\SAP\jCO.jar;
"%CROSSWORLDS%"\ODA\SAP\sapjco.jar
set JCLASSES=%AGENT%;%JCO_JAR%;%CWLIB%;%WBIA%
echo classpath = %JCLASSES%
javac -classpath %JCLASSES% %1
endlocal
pause
```

Part 5. RFC Server module

Chapter 17. Overview of the RFC Server Module

This chapter introduces the RFC Server Module of the Adapter Guide for mySAP.com (R/3 V.3.x). The RFC Server Module enables the integration broker to receive business objects from SAP applications that support RFC calls. It supports all SAP applications that use RFC-enabled functions by acting as a server to those applications.

This chapter contains the following sections:

RFC Server Module components

The RFC Server Module is a connector module written in Java that supports RFC calls directly from an SAP application. It extends the Vision Connector Framework by implementing the VisionConnectorAgent class. The RFC Server Module uses the SAP RFC libraries that are written in Java and C, which enables external programs to communicate with an SAP application.

Figure 25 on page 184 illustrates the overall architecture of the RFC Server Module. The RFC Server Module is made up of the connector framework, the connector's application-specific component for RFC Server, RFC Server-specific business object handlers, listener threads, and the SAP RFC Library.

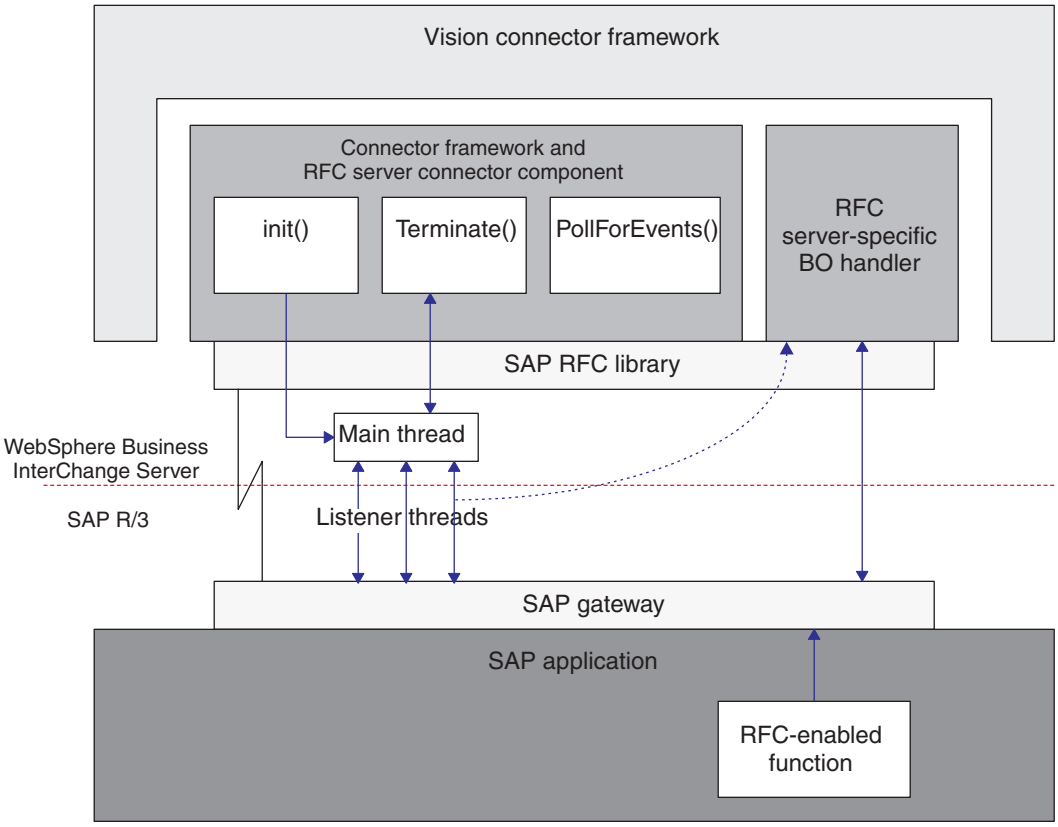


Figure 25. RFC Server Module architecture

The RFC Server Module components:

- Spawn listener threads that open handles to the SAP application using the SAP RFC library and the SAP Gateway. Each listener thread opens a single handle to the SAP application.
- Process requests from RFC-enabled functions in the SAP application.
- Terminate connections to the SAP application.

Listener threads

Listener threads handle all of the RFC calls between the RFC Server Module and the SAP application. When the connector starts up, the `init()` method creates a main thread that spawns a configurable number of listener threads. Each listener thread opens a handle to the SAP Gateway.

The listener threads:

- Register with the SAP Gateway using a program identifier.
- Identify to the SAP Gateway the RFC-enabled functions that they support.
- Use the first available thread to pick up an event from a supported RFC-enabled function.
- Instantiate an RFC Server-specific business object handler based on the Server verb in the corresponding business object, and then retrieve the event data from the SAP Gateway.
- Populate business objects with RFC event data, and then convert returned business object data to RFC event data.
- Return a response to the RFC-enabled function through the SAP Gateway.

Note: A thread listens continuously in a synchronous manner for events from RFC-enabled functions that it supports.

RFC Server-specific business object handlers

The RFC Server-specific business object handlers are unique to each RFC-enabled function in the SAP application. Each business object handler is instantiated by a listener thread and invokes an associated business object.

Because the RFC Server Module acts as a server to the SAP application, it "pushes" or sends events from the SAP application to the integration broker. This behavior is very different from other modules, which poll the application for events. Because of this difference, RFC Server-specific business object handlers perform different tasks from other business object handlers.

Once instantiated, the RFC Server-specific business object handler:

- Retrieves the RFC event data and populates the associated WebSphere business object for SAP.
- Passes the business object to the integration broker and receives a business object in return.

  The business object handler uses the application-specific information of the business object's Server verb to determine which collaboration should process the business object data.
  - When WebSphere InterChange Server is the integration broker, the business object's Server verb must specify a valid collaboration. Because a collaboration cannot explicitly subscribe to an event that is pushed to the connector, the RFC Server-specific business object handler must determine the appropriate collaboration, and then instantiate it
  - When a WebSphere message broker is the integration broker, the business object's Server verb must contain a dummy value for the collaboration.
- Converts the returned business object data back to RFC event data.
- Returns the RFC event data back to the SAP application.

How the RFC Server Module works

The RFC Server Module implements the `init()`, `terminate()`, `pollForEvents()`, and `process()` methods.

This section describes:

- "Initialization and termination" on page 185
- "Business object processing" on page 186
- "Supporting RFC-enabled functions" on page 187

Initialization and termination

The `init()` method creates a main thread that spawns a configurable number of listener threads which open a handle to the SAP Gateway. If the connector fails to initialize, it terminates using the `terminate()` method. The connector terminates by disconnecting the connection to the SAP Gateway.

During the initialization process, the RFC Server Module registers with the SAP Gateway using a specified Program ID. This Program ID must be set using the RfcProgramID connector configuration property and set up as a TCP/IP port in the

SAP application. For more information on setting up a TCP/IP port, see "Registering the RFC Server Module with the SAP gateway" on page 189.

Business object processing

All processing of WebSphere business objects for the RFC Server Module is initiated by an RFC-enabled function in an SAP application. In the RFC Server Module, an RFC Server-specific business object handler supports only one RFC-enabled function; therefore, for each supported function in the SAP application, you must have an associated RFC Server-specific business object handler. In addition, you must have an associated business object for each RFC Server-specific business object handler.

Figure 26 illustrates business object processing for the RFC Server Module.

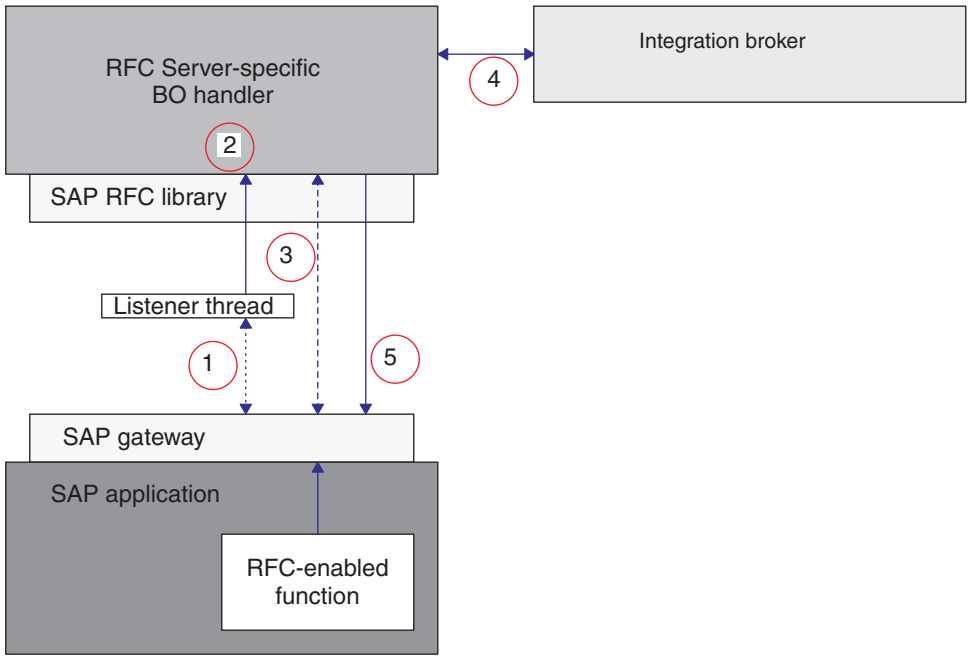


Figure 26. Business object processing

Business object processing for the RFC Server Module executes in the following manner:

1. A listener thread picks up a subscribed event from the SAP Gateway and matches the name of the corresponding RFC-enabled function with an RFC Server-specific business object handler.
2. The listener thread instantiates the appropriate RFC Server-specific business object handler based on data from the RFC event on the SAP Gateway, and then creates an instance of the corresponding business object.
3. The RFC Server-specific business object handler retrieves the RFC interface data from the SAP Gateway and populates the WebSphere business object for SAP.
4. The RFC Server-specific business object handler passes the business object to the integration broker. In the RFC Server Module, since SAP makes the synchronization calls, when a WebSphere message broker is the integration broker, the RFC Server Module uses SynchronousRequestQueue and SynchronousResponseQueue to communicate with the WebSphere message broker.

5. The business object handler receives the returned business object from the integration broker, converts it back to the RFC interface, and then returns it to the SAP Gateway.

The RFC Server Module uses the SAP Gateway to maintain the processing order of events and to maintain the status of events. Since the listener threads make synchronous calls, an event must return to the SAP Gateway before it can be considered successfully processed.

Note: If an RFC-enabled module has a Return Structure or Return Table, the connector checks for the message types `A` (abort) and `E` (error) to determine if the event processed successfully. A message type `A` or `E` indicates that the event failed to process. If an RFC-enabled function module does not have a Return Structure or Return Table, you must implement your own error handling. The error message or messages, within the structure or table, are returned in the return status descriptor.

Supporting RFC-enabled functions

The development environment IBM WebSphere Business Integration Adapter for mySAP.comincludes a utility, SAPODA, that generates business object definitions based on an RFC-enabled function. SAPODA interprets the interface of an RFC-enabled function, maps its interface parameters to the business object attributes, and adds the application-specific information for each attribute.

For each business object definition, you must generate an associated RFC Server-specific business object handler, which invokes the corresponding business object. For more information on developing business objects and RFC Server-specific business object handlers, see Chapter 19, “Developing business objects for the RFC Server Module,” on page 191.

Note: Some RFC-enabled functions do not have single field parameters that correspond to simple attributes in the WebSphere business object. The connector requires every top-level business object to have a simple attribute that serves as the key attribute. Therefore, when generating a business object and business object handler from a RFC-enabled function without a single field parameter, SAPODA creates a key attribute named Dummy_key in the top-level business object, marks it as the key attribute, and adds `dummy_key` as the application-specific information of this attribute. Dummy_key provides the connector with a key attribute so that it can process the business object. However, the connector ignores the value of the Dummy_key attribute when modifying application data.

Triggering an event

To trigger an event for the RFC Server Module the RFC destination must be specified for the remote function call. The remote function call can be executed in two ways: programmatically and using transaction SE37. Programmatically, the variation of the CALL FUNCTION command that specifies a destination must be used. The value to specify for the destination is the one that is created to register the RFC Server Module. See section “Registering the RFC Server Module with the SAP gateway” for more information. Using transaction SE37, the RFC target system must match the RFC destination. See section “Registering the RFC Server Module with the SAP gateway” for more information on creating and registering a RFC destination for the RFC Server Module.

Chapter 18. Configuring the RFC Server Module

This chapter describes the configuration of the RFC Server Module and assumes that all of the necessary files were installed when the Adapter Guide for mySAP.com (R/3 V.3.x) was installed. For more information on installing the connector, see Chapter 2, "Installing and configuring the connector," on page 11.

This chapter contains the following sections:

- "RFC Server Module directories and files"
- "RFC Server Module configuration properties"
- "Registering the RFC Server Module with the SAP gateway"

RFC Server Module directories and files

The RFC Server Module directory and files are contained in the `\connectors\SAP\` directory. Table 38 lists the directory and file used by the RFC Server Module.

Table 38. RFC Server Module directory and file

Directory/filename	Description
`\bapi\server`	Directory containing the runtime files for the connector. All RFC Server-specific BO Handler class files must be copied into this directory.
`CWSAP.jar`	Connector class file

RFC Server Module configuration properties

You must configure the RFC Server Module before it can start operating. To configure the RFC Server Module, set the standard and connector-specific connector configuration properties. For more information on configuring the connector configuration properties, see "Configuring the connector" on page 17 and Appendix B, "Standard configuration properties for connectors," on page 241.

Registering the RFC Server Module with the SAP gateway

During initialization, the RFC Server Module registers with the SAP Gateway. It uses the value set for the `RfcProgramId` connector-specific configuration property. This value must match the value set in the SAP application. You must configure the SAP application so that the RFC Server Module can create a handle to it.

To register the RFC Server Module as an RFC destination:

1. In the SAP application, go to transaction SM59.
2. Expand the TCP/IP connections directory.
3. Click Create (F8).
4. In the RFC destination field, enter the name of the RFC destination system. It is recommended that you use `RFCSERVER`.
5. Set the connection type to T (Start an external program via TCP/IP).
6. Enter a description for the new RFC destination, and then click Save.
7. Click the Registration button for the Activation Type.

8. Set the Program ID. It is recommended that you use the same value as the RFC destination (RFCSERVER), and then click Enter.

Important: Ensure that the connector-specific configuration property RfcProgramID is set to the same value as the Program ID value in the SAP application. If the values do not match, business object processing will fail.

Chapter 19. Developing business objects for the RFC Server Module

This chapter describes business objects and business object handlers required for the RFC Server Module. It provides background information and discusses how the business object generation utility, SAPODA, generates the definitions.The chapter assumes you are familiar with how the connector processes business objects. For more information on business object processing in the RFC Server Module, see Chapter 17, "Overview of the RFC Server Module," on page 183.

This chapter contains the following sections:

Note: Once you have created business objects and RFC Server-specific business object handlers, you must make sure that you register the RFC Server Module with the SAP Gateway. For more information, see "Registering the RFC Server Module with the SAP gateway" on page 189..

Background information

Business object development for the RFC Server Module consists of creating an application-specific business object definition and an associated RFC Server-specific business object handler for each RFC-enabled function that you want to support. Because SAPODA uses the SAP application's native definitions as a template when generating definitions for each of these, it is recommended that you use SAPODA to generate these definitions.

Note: SAP supports many methods that can be mapped to the standard verbs (Create, Update, Delete, and Retrieve) that the connector supports. You can develop business objects and RFC Server-specific business object handlers to support any method used by RFC-enabled functions.

Business object naming conventions

An RFC-enabled function interface consists of importing, exporting, and table parameters, where:

- Importing parameters are passed to the RFC-enabled function
- Exporting parameters are returned from the RFC-enabled function
- Table parameters are passed in either direction

Some RFC-enabled functions may not have all of the types of parameters. For example, an RFC-enabled function may have importing and table parameters only.

SAPODA automatically maps the RFC-enabled function importing, exporting, and table parameters to IBM WebSphere attributes as described in Table 39..

Table 39. Naming conventions: WebSphere Business Objects for SAP

Business object	Rfc-enabled function interface
Top-level business object	*BOprefix_FunctionName***Note**: The illustrations in this chapter use `SAP_` or `sap_` as the business object prefix. You can specify your own meaningful prefix when you create your business object definitions.
Attribute	*Field Description or Field Name*
Child business object	*BOprefix_FunctionParameterName*

SAPODA guarantees that all attribute names in the business object definition are unique. If an RFC-enabled function has multiple parameters with the same field description, SAPODA adds a counter as the suffix to the generated attribute name.

When naming an attribute from a RFC-enabled function parameter, SAPODA prepends a string to the attribute name when the changed attribute name:

- Begins with a digit—prepends `A_`
- Begins with the underscore character (_)—prepends `A`

Important: You can modify the attribute names at any time after you generate the business object definition. However, when you modify an attribute name, do not modify the application-specific information. The connector uses this information to identify the RFC-enabled function parameter to which the attribute corresponds. For more information on the application-specific information, see "AppSpecificInfo for Attributes" on page 197.

Business object structure

The connector uses an RFC Server-specific business object handler to map each business object attribute to an RFC-enabled function's parameter. The connector, each business object, and each RFC Server-specific business object handler are metadata-driven. The application-specific information provided in the metadata of each business object and business object handler allows you to add connector support for a new business object and its handler without modifying connector code. Instead:

- The connector uses the verb application-specific information of the top-level business object to instantiate the appropriate RFC Server-specific business object handler.

  Important: The RFC Server Module differs from other modules in that it does not poll SAP for events. Instead, SAP pushes event data to the connector. Because this module does not use standard polling procedures, the RFC Server-specific business object handler checks every business object that represents an event for the name of the collaboration that will process it. When the WebSphere InterChange Server is the integration broker, the RFC Server-specific business object handler uses the value obtained to instantiate the appropriate collaboration.
- The business object handler uses the attribute application-specific information of each business object to map between each attribute and its parameter.

Each RFC Server-specific business object handler supports both single- and multiple-cardinality relationships between business objects.

A WebSphere business object based on an RFC-enabled function can contain no more than two levels of hierarchy. Therefore, all simple parameters correspond to attributes of the top-level business object, and structure and table parameters correspond to child business objects.

Table 40. Correspondence between RFC-enabled functions and business objects

RFC-enabled function interface parameter	**WebSphere business object for SAP**
Simple field	Attribute of the top-level business object
Structure	Single-cardinality child business object
Table	Multiple-cardinality child business objects

Note: Importing and exporting parameters can be simple field or structure parameters.

Figure 27 illustrates the association between a WebSphere business object and an RFC-enabled function, in this instance, a BAPI. The figure illustrates a fragment of a user-defined sap_bapi_po_create business object, which corresponds to the `BAPI_PO_CREATE` BAPI.

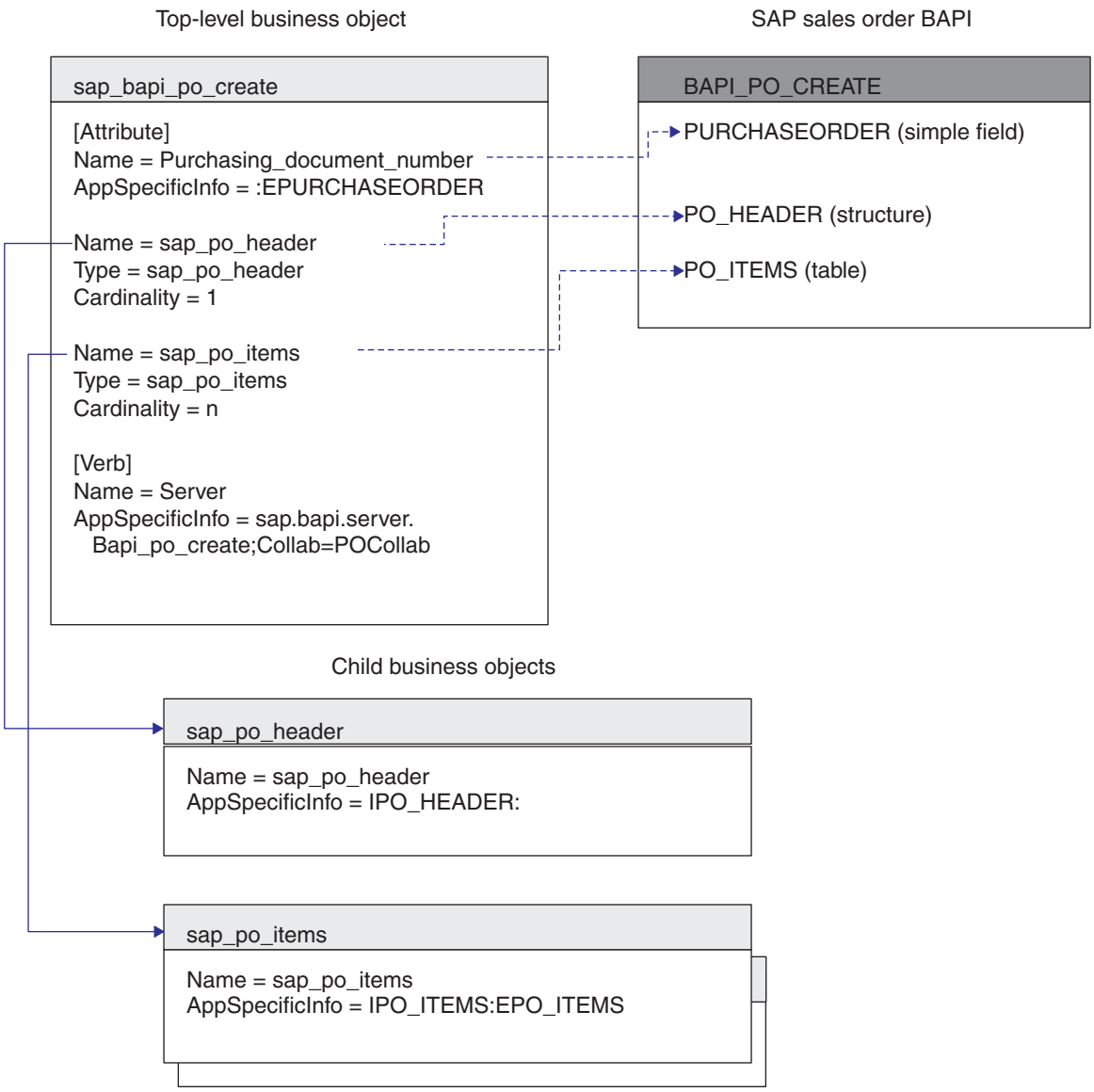


Figure 27. Mapping between a business object and a BAPI

Supported verbs

The RFC Server Module supports the standard verbs (Create, Update, Delete, and Retrieve) used by the WebSphere business integration system. For each supported verb, an RFC-enabled function can have an associated method. Most RFC-enabled functions support one of the following operations: create, retrieve, update, and delete.

Business object attribute properties

The properties of the attributes of a top-level business object differ depending on whether the attribute represents a simple value, or a child or an array of child business objects.

- Table 41 lists and describes the properties of simple attributes of a top-level business object.
- Table 42 lists and describes the attributes that represent a child or array of child business objects.

SAPODA generates the attribute properties as described in each table.

Table 41. Simple attributes: Top-Level business object

Property name	Description
Name	Derived from the description or name of the RFC-enabled function parameter. SAPODA replaces special characters (such as periods, slashes, and spaces) with underscores.
Type	Specifies the type of data. SAPODA sets the value to `String`.
MaxLength	Specifies the field length of the RFC-enabled function parameter.
IsKey	Specifies whether the attribute is the key. The first simple attribute of a business object defaults to the key attribute. The connector does not support using an attribute that represents a child business object or an array of a child business objects as a key attribute. Therefore, if the function provides only structure and table parameters, you must insert a simple attribute as the first attribute. SAPODA inserts the Dummy_key attribute as the first attribute, marks it as the key attribute, and sets appropriate values. Do not modify those values. For more information, see "Supporting BAPIs" on page 166.
IsForeignKey	SAPODA sets the value to `false`.
IsRequired	Specifies whether an attribute must contain a value. SAPODA sets the value to `false`.
AppSpecificInfo	Contains the name of the RFC-enabled function that corresponds to the associated attribute. The format is: `I`*`RFCFunctionParameterName`*`:E`*`RFCFunctionParameterName`* For more information on the application-specific information, see "Business object application-specific information" on page 196.
Default Value	Specifies the value to assign to this attribute if there is no run-time value. SAPODA does not set a value for this property.

Table 42 lists and describes the attributes that represent a child or an array of child business objects. SAPODA generates the properties described in the table below.

Table 42. Properties of an attribute that represents a child or children

Property Name	Description
Name	The value is the name of the structure or table parameter name. The format is: `BOprefix_FunctionParameterName`
Type	The value is the type of child business object; in other words, the type is `BOprefix_FunctionParameterName`
ContainedObjectVersion	SAPODA sets the value to `1.0.0`.
Relationship	SAPODA sets the value to `containment`.
IsKey	SAPODA sets the value to `false`.
IsForeignKey	SAPODA sets the value to `false`.
IsRequired	Specifies whether an attribute must contain a value. SAPODA sets the value to `false`.

Table 42. Properties of an attribute that represents a child or children (continued)

Property Name	Description
AppSpecificInfo	Contains the name of the RFC-enabled function parameter that corresponds to the associated attribute. The format is: `I`*`FieldName`*`:E`*`FieldName`* For more information on the application-specific information, see "Business object application-specific information."
Cardinality	Structure parameters have single cardinality (1) and table parameters have multiple cardinality (`n`).

Initializing attribute values

Every field in SAP has an initial value, as listed in Table 43. When the connector receives an event, the RFC Server-specific business object handler moves these values from each SAP field to its corresponding business object attribute. The business object handler retains initial values from SAP with one exception: the character data type. The business object handler converts a space in the SAP field to `CxIgnore` in the business object attribute. If you want any other value to be converted to `CxIgnore`, the component that creates the business object must perform the conversion. For example, when the WebSphere InterChange Server is the integration broker, modify the map to handle this conversion.

Table 43. Initial field values in SAP

Data type	Description	Initial Value Set by business object handler
`C`	Character	`space`
`N`	Numeric string	*000...*
`D`	Date (YYYMMDD)	`00000000`
`T`	Time (HHMMSS)	`000000`
`X`	Byte (hexadecimal)	`X00`
`I`	Integer	`0`
`P`	Packed number	`0`
`F`	Floating point number	`0.0`

Business object application-specific information

Application-specific information in business object definitions provides the RFC Server Module with application-dependent instructions on how to process business objects. These instructions are specified at the business-object level, at the attribute level (both for simple attributes and for attributes that represent a child or array of child business objects), and for verbs.

AppSpecificInfo for the server verb of the top-level business object

The connector uses the value of the Server verb's application-specific information in the top-level business object to call the appropriate RFC Server-specific business object handler and to determine the destination collaboration for event processing. The value of the `AppSpecificInfo` property for the Server verb specifies:

- the package and classname for the RFC Server-specific business object handler
- the destination collaboration

The format is as follows:

```
AppSpecificInfo = bapi.server.BOHandler;Collab=CollaborationName
```

where BOHandler is the name of the class and *CollaborationName* is the name of the destination collaboration.

SAPODA automatically adds the application-specific information for the Server verb in top-level business object. For the value of the business object handler's classname, it uses the name of the RFC-enabled function. It does not provide a value for the collaboration name parameter. Therefore, you must manually add the name of the collaboration.

Note: There is a one-to-one relationship between the WebSphere business object for SAP and the RFC Server-specific business object handler. The business object handler class files must exist in the \connectors\SAP\bapi\server directory.

Important: You must include the value server before the business object handler name to identify that the RFC Server-specific business object handler acts as a server.

For example if you are supporting the BAPI_PO_CREATE RFC-enabled function and the destination collaboration is called POCollab, then the verb application-specific information is as follows:

```
AppSpecificInfo =bapi.server.Bapi_po_create;Collab=POCollab
```

AppSpecificInfo for Attributes

The connector uses the value of an attribute's application-specific information to determine which importing, exporting, and table parameters to use. The value of this property contains the prefix I (for importing parameters) or E (for importing parameters). The prefix indicates whether the attribute value is used to pass data into or out from the SAP application.

Because structure parameters can be either importing or exporting, they use either an I or an E before the parameter value. Because table parameters can pass data to and return data from a RFC-enabled function, they can have both I and E parameter values.

Important: Always use a colon (**:**) separator when you specify parameter values with I and E. If specifying only an importing value, the colon must follow the value. If specifying only an exporting value, the colon must precede the value. If specifying both values, the colon follows the importing value and precedes the exporting value.

Figure 28 illustrates the mapping between a business object and an example RFC-enabled function named BAPI_EXAMPLE. In the example, the simple attributes (Attribute_1, Attribute_2, and Attribute_3) specify only an importing or exporting parameter. The attribute that represents a child business object (Child_1) maps to an exporting structure parameter. The attribute that represents an array of child business objects (Child_2) maps to a table parameter.

Each child business object has a simple attribute that maps to a field of the corresponding structure or table (Attribute_11 and Attribute_14, respectively). You can find these fields by looking at the details of the BAPI.

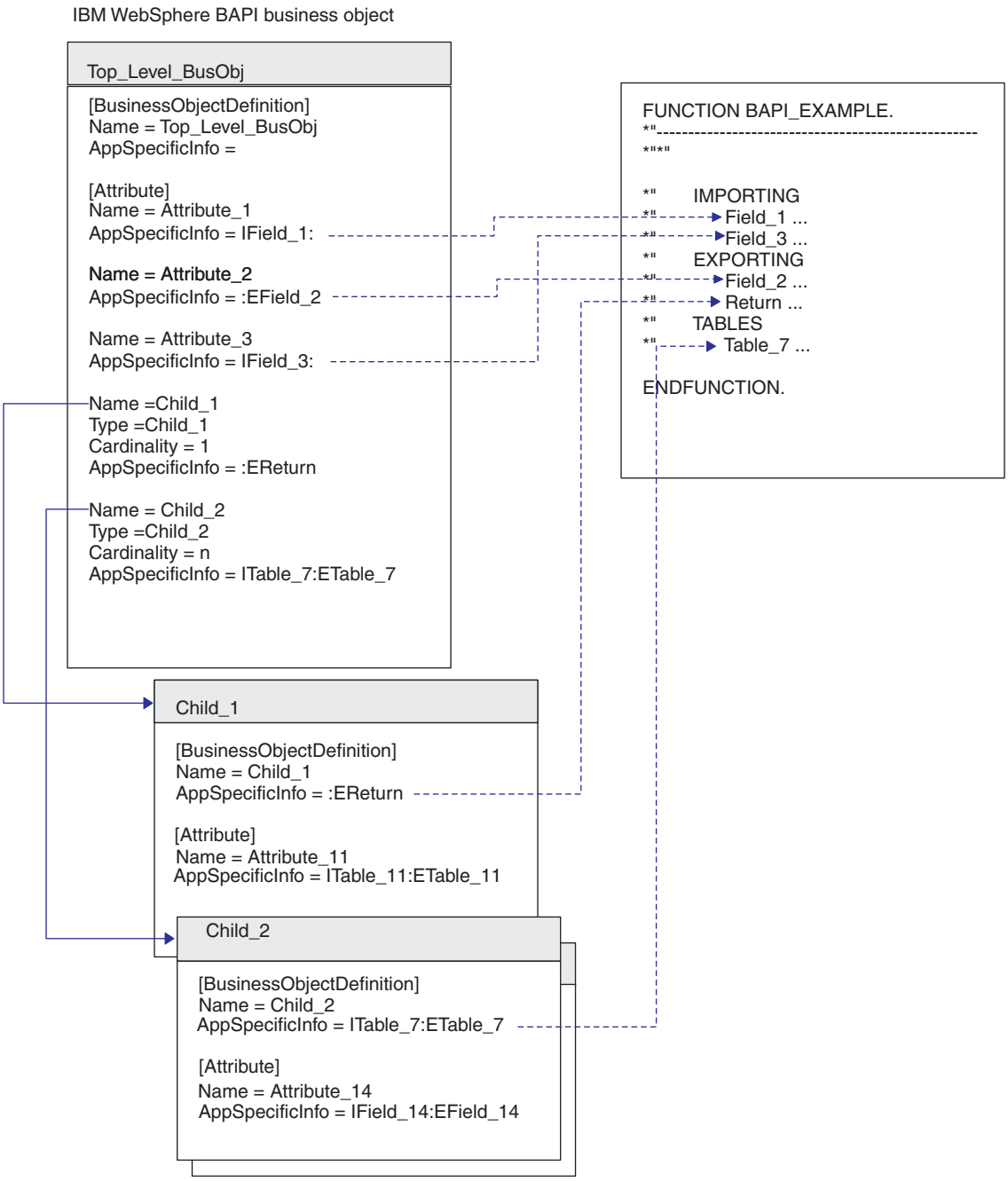


Figure 28. Mapping between a business object and an example BAPI

Table 44 identifies the format of the application-specific information for specific kinds of attributes.

Table 44. AppSpecificInfo format for specific kinds of attributes

AppSpecificInfo format	Attribute type
I*ParameterName*:E*ParameterName*	Simple
I*TableName*:E*TableName*	Represents a child business object mapped to a table parameter
I*StructureName*:E*StructureName*	Represents a child business object mapped to a structure parameter
I*FieldName*:E*FieldName*	Represents an attribute of a child business object mapped to a field in a table or structure parameter

SAPODA automatically generates the appropriate application-specific information for your business object definition. It is recommended that you do not change the parameter names of the generated application-specific information.

Using generated business objects and business object handlers

Use SAPODA to generate RFC-enabled function-specific business object definitions and RFC Server-specific business object handlers for each RFC-enabled function you want to support. You can use the generated files with minimal modifications.

The only edit you must make is specifying the name of the destination collaboration in the verb application-specific information of the Server verb.

- When the WebSphere InterChange Server is the integration broker, this information is required because a collaboration cannot explicitly subscribe to an event that is pushed to the connector. Therefore, the RFC Server-specific business object handler must determine the appropriate destination collaboration from the business object's metadata, and then instantiate the collaboration.

Important: If the RFC-enabled function that you are using does not contain a simple field attribute, and SAPODA has created a Dummy_key attribute as the key attribute, do not modify the values of this attribute.

After the business object definition and its corresponding RFC Server-specific business object handler are generated, you must add the business object definition to your WebSphere business integration system's runtime environment.

- Use Business Object Designer to load the business object definition into your repository.

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

- Use a system command to copy the RFC Server-specific business object handler files to the following directory under the product directory:

  ```
  \connectors\SAP\bapi\server
  ```

The RFC Server-specific business object handler files are:

- *RFC-EnabledFunctionName*`.java`
- *RFC-EnabledFunctionName*`.class`

For example, given the `BAPI_PO_CREATE` RFC-enabled function and a user-specified prefix of `sap_`, SAPODA generates the following:

- `sap_bapi_po_create` (business object definition that includes all child business objects)
- `Bapi_po_create.java`
- `Bapi_po_create.class`

Important: You can modify the name of the generated business object as well as the name of its child business objects. To do so, you must edit the definition as a text file rather than in Business Object Designer. If you do change a business object's name, ensure that you also modify all references to the names that you change. Also, if you modify the names of the generated `.class` file for the business object handler, you

must maintain the changes for the Server verb application-specific information for the associated business object.

Note: For RFC-enabled ABAP functions and BAPIs that are developed in a development namespace, SAPODA removes or replaces "/" characters in the function name with "_" when naming the business object definition, .java, and .class files. SAPODA removes the "/" character only when it is the first character of the name. Although the definition name or file name does not contain this character, the code still accurately calls the specified function with its proper name containing the "/" characters. Also, when a function name begins with a digit, SAPODA prepends the name with the string Rfm_.

Tips and tricks

The following are tips and tricks for developing business objects and RFC Server-specific business object handlers.

- "Multiple business objects contain the same return business object"
- "Generated business object definition contains unnecessary attributes and child business objects" on page 201
- "Generated business object names are too long or fail your naming conventions" on page 201
- "Generated AppSpecificInfo for table parameters specify unnecessary parameters" on page 201

Multiple business objects contain the same return business object

Most RFC-enabled functions use the same name for the return object. When SAPODA generates a business object definition, it creates a child business object to represent this return object. If multiple business object definitions contain an identically named child business object, you can add the definition for child business object into the repository only once.

To enable multiple business objects to contain the return business object, you must modify the name of the return business object to be unique for each business object.

To rename the return business object, modify the definition of each business object definition that contains it. The definition of the child business object is contained in the same definition file as its parent.

To rename the child, do the following:

1. Open the definition file for the top-level business object in a text editor.
2. Locate the definition of the BOprefix_return child business object.
3. Change the child's name to be unique. For example, append a number to the text (sap_return_2).
4. Change all references in the definition to refer to the newly named child. For example, change the value of the Type property for every attribute that represents the child business object.
5. Save the changed definition file.
6. Use Business Object Designer to load the newly named child business object into the repository.

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

Generated business object definition contains unnecessary attributes and child business objects

SAPODA interprets all RFC-enabled function interface parameters and, for each one, it creates a corresponding WebSphere business object attribute or child business object. To increase performance of business object processing, remove all unneeded attributes and business objects from the business object definition.

Note: SAPODA facilitates graphically removing all optional attributes and child business objects before definition generation. For more information, see "Provide additional information" on page 303.

To increase performance of business object processing, you can also remove all unneeded importing and exporting table parameter values from the application-specific information.

After definition generation, you can use Business Object Designer to manually edit the business object definition if you require other changes. However, be careful that you remove only attributes that you absolutely will not be using.

Generated business object names are too long or fail your naming conventions

SAPODA uses the name of the RFC-enabled function module to name the generated business object. You can use a text editor to modify a business object's name.

Important: If you do change the name, ensure that you modify all references to the name as well. However, do not modify the parameter names of the generated application-specific information.

To change a generated business object's name:

1. Save the definition to a file.
2. Use a text editor to shorten or change the name.
3. Use Business Object Designer to copy the newly named child business object into the repository.

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

Generated AppSpecificInfo for table parameters specify unnecessary parameters

Table parameters can be both importing and exporting parameters. If you do not require importing or exporting of values for a table parameter, you can remove it from the application-specific information.

For example, for a create operation, if you do not need to return the table data from the SAP application after the create operation has completed, you can remove the exporting parameter value (such as E*table name*).

For a retrieve operation, you do not need to specify any importing table parameters. Therefore, you can remove the importing parameter value (such as I*table name*).

Note: You must remove the unrequired value from the `AppSpecificInfo` of the attribute in the parent that represents the child as well as from the `AppSpecificInfo` at the business-object level of the child business object. Do not remove the colon (`:`).

For example, to remove the `ETable_7` exporting parameter in Figure 28, you would do the following:

1. In the Child_2 attribute of the Top_Level_BusObj business object, change the attribute's AppSpecificInfo value to:

   ```
   ITable_7:
   ```

2. In the AppSpecificInfo at the business-object level of the Child_2 business object, change the value to:

   ```
   ITable_7:
   ```

3. In the AppSpecificInfo for each attribute of the child business object, using Attribute_14 as an example, change the value to:

   ```
   IField_14:
   ```

Part 6. Hierarchical Dynamic Retrieve module

Chapter 20. Overview of the Hierarchical Dynamic Retrieve Module

This chapter describes the Hierarchical Dynamic Retrieve moduleIBM WebSphere Business Integration Adapter for mySAP.com. The Hierarchical Dynamic Retrieve Module processes hierarchical or flat business objects. To process these requests, the connector retrieves data from the SAP R/3 application version 3.x.

This chapter contains the following sections:

Hierarchical Dynamic Retrieve Module components

The Hierarchical Dynamic Retrieve Module is written in Java and extends the vision connector framework. Because the module does not have its own application-specific component, it uses the application-specific component for BAPI. Therefore, the module consists of the connector framework, the application-specific component for BAPI, the DynRetBOH business object handler, and the SAP RFC libraries. SAP delivers the RFC libraries in Java and C. The connector is delivered and runs as a Java archive (JAR) file.

Figure 29 on page 206 illustrates the architecture of the Hierarchical Dynamic Retrieve Module.

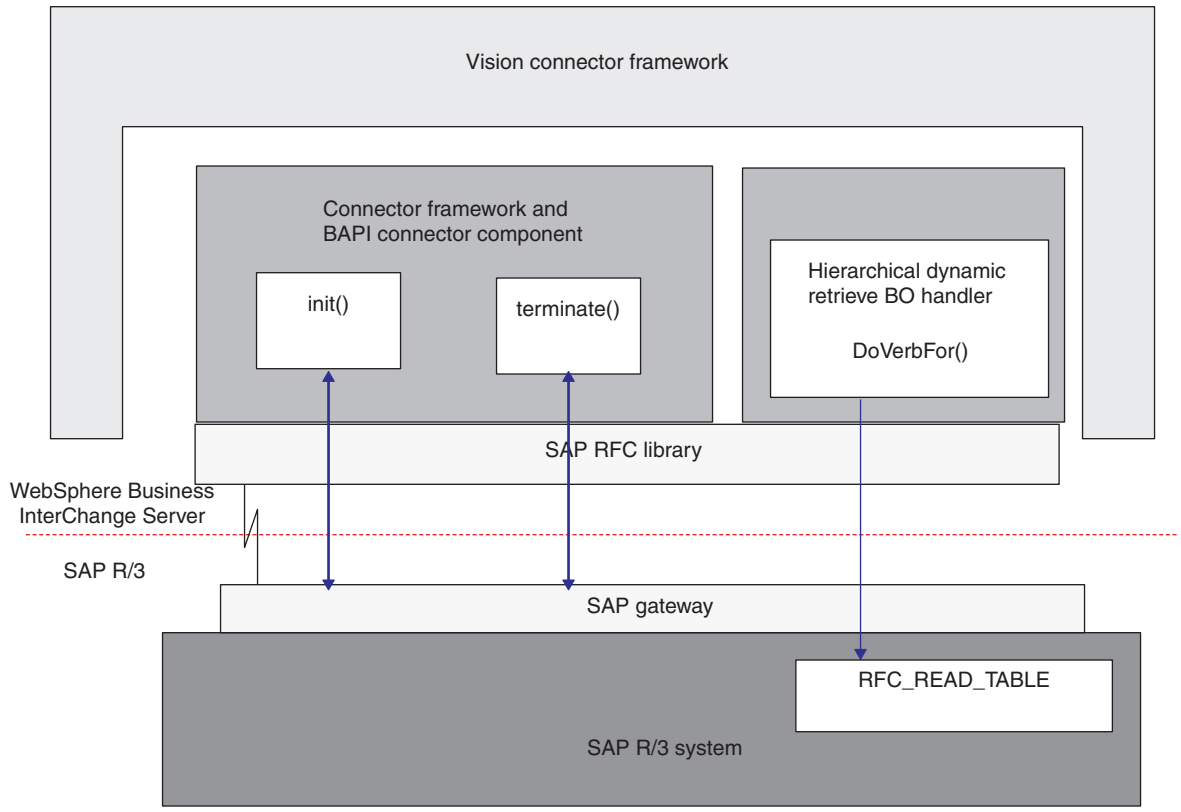


Figure 29. Hierarchical Dynamic Retrieve Module architecture

How the connector works

The connector gets a business object's processing information from metadata specified in the business object rather than from information hard-coded into the connector. To obtain processing information from the business object, the connector makes assumptions about the following:

- The business object structure
- The relationships between parent and child business objects
- The possible database representations of business objects

For information, see "Processing business objects" on page 6, and Chapter 22, "Developing business objects for the Hierarchical Dynamic Retrieve Module," on page 211.

When the connector receives a request from the integration broker to perform an application operation, it obtains processing information from the verb specified for the top-level business object.

The connector processes hierarchical business objects recursively; that is, it performs the same steps for each child business object until it has processed all individual business objects.

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

any child business objects it might contain or that contain it. The term **top-level** business object refers to the individual business object at the top of the hierarchy that does not itself have a parent business object.

When the integration broker sends a hierarchical business object with a Retrieve verb, the connector attempts to return a business object to the integration broker that exactly matches the current database representation of that business object. In other words, the value of each simple attribute of every individual business object that the connector returns matches the value of its corresponding field in the database. Also, the number of individual business objects in each array of the returned business object match the number of children in the database for that array (unless the application-specific information limits the children to a subset).

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

Chapter 21. Configuring the Hierarchical Dynamic Retrieve Module

This chapter describes the configuration of the Hierarchical Dynamic Retrieve Module of the Adapter Guide for mySAP.com (R/3 V.3.x). The SAP connector should be installed before performing the configuration tasks described in this chapter. For more information on installing the connector, see Chapter 2, "Installing and configuring the connector," on page 11.

This chapter contains the following sections:

- "Hierarchical Dynamic Retrieve Module directories and files"
- "Hierarchical Dynamic Retrieve Module configuration properties"

Hierarchical Dynamic Retrieve Module directories and files

Table 45 lists the directories and files used by the Hierarchical Dynamic Retrieve Module.

Table 45. Hierarchical Dynamic Retrieve Module directories and files

Filename	Description
`CWSAP.jar`	Connector class file

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.

Hierarchical Dynamic Retrieve Module configuration properties

Before you can run the Hierarchical Dynamic Retrieve Module, you must set the standard and connector-specific configuration properties. At a minimum, you must add the class name for the BAPI Module to the module's property. The classname is `sap.bapimodule.VBapiAgent`.

For more information on configuring the connector configuration properties, see "Configuring the connector" on page 17 and Appendix B, "Standard configuration properties for connectors," on page 241.

Chapter 22. Developing business objects for the Hierarchical Dynamic Retrieve Module

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

In addition to providing background information on business objects and their processing, the chapter describes how to develop business objects for the Hierarchical Dynamic Retrieve Module using:

- SAPODA (an Object Discovery Agent)— generates business object definitions from tables you specify graphically. This utility is most useful for creating individual business object definitions rather than hierarchical business object definitions.
- Advanced Outbound Wizard—records and interprets your actions as you step through an SAP display transaction. As it generates the business object definition, it automatically defines the relationships between parent and child business objects.

For a description of the Hierarchical Dynamic Retrieve Module, see Chapter 20, "Overview of the Hierarchical Dynamic Retrieve Module," on page 205.

This chapter contains the following sections:

Business object development utilities

Business object development for the Hierarchical Dynamic Retrieve Module requires you to create an application-specific business object definition for each type of object you want the connector to handle. The IBM WebSphere Business Integration Adapter for mySAP.com (SAP R/3 Version 4.x) includes the following:

- The vDynRetBOH business object handler, which the connector uses to retrieve data from the application
- SAPODA
- Advanced Outbound Wizard

Although you can use Business Object Designer or a text editor to create business object definitions for the connector, it is recommended that you initially use SAPODA or Advanced Outbound Wizard, because they use the SAP application's native definitions as a template.

For more information on using Advanced Outbound Wizard, see "Generating business objects: Advanced Outbound Wizard" on page 225.

Business object names

SAPODA and Advanced Outbound Wizard guarantee that all attribute names in the business object definition are unique. They derive the names from SAP's data dictionary by appending the field's name and description. When naming an attribute from an SAP table, SAPODA prepends a string to the attribute name when the changed attribute name:

- Begins with a digit—prepends `A_`
- Begins with the underscore character (_)—prepends `A`

Attention: The attribute names can be modified at any time after the business object has been generated. Changing the business object's name or attribute names does not affect the processing of the business object. However, changing the application-specific information **does** affect the processing of the business object, because the application-specific information identifies the SAP table and column to which the attribute corresponds.

For more information on the application-specific information, see "Business object application-specific information" on page 221. For information on using the wizard, see "Generating business objects: Advanced Outbound Wizard" on page 225.

Business object structure

The connector assumes that every individual business object is represented by one or more database tables, and that each **simple attribute** (that is, an attribute that represents a single value, such as a `String` or `Integer` or `Date`) within the business object is represented by a column in one of those tables. The following situations are valid:

- The database tables might have more columns than the corresponding individual business object has simple attributes (that is, some columns in the database are not represented in the business object). Include in your design only those columns needed for the business object processing
- The individual business object might have more simple attributes than the corresponding database tables have columns (that is, some attributes in the business object are not represented in the database). The attributes that do not have a representation in the database have no application-specific information
- Due to a restriction in the SAP API, the total number of bytes for all of the desired columns in each table represented by a single a business object cannot exceed 512 For more information, see "Handling long data rows" on page 216
- Due to restrictions in the SAP API, runtime HDR modules may not be able to parse some of the non-character based datatypes. Please refer to "Troubleshooting the Hierarchical Dynamic Retrieve Module" on page 287

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

A hierarchical business object has attributes that represent a single child business object, an array of child business objects, or a combination of both. In turn, each child business object can contain a single child business object or an array of business objects, and so on.

Business object relationships

The Cardinality property of the attribute that represents the child or array determines the type of relationship between parent and child:

- A **single-cardinality relationship** occurs when the attribute in the parent business object represents a child business object with cardinality 1.
- A **multiple-cardinality relationship** occurs when an attribute in the parent business object represents an array of child business objects with cardinality `n`.

The connector does not process a single-cardinality relationship differently from a multiple-cardinality relationship. However, there is a structural difference in foreign-key relationships when database tables have single-cardinality or multiple-cardinality relationships. This difference is important when Advanced Outbound Wizard generates a business object definition from an SAP Display Transaction:

- In a single-cardinality relationship, the foreign key is determined by the primary key in the child referencing a **non- key** attribute in the parent as its foreign key. Each child has at least one simple attribute that references a non-primary key attribute in its parent as a foreign key. Figure 30 on page 214 provides an example.
- In a multiple-cardinality relationship, the foreign key is determined by the primary key in the child referencing the **primary key** attribute in the parent. Each child has at least one simple attribute that contains the parent's primary key as a foreign key. The child has as many foreign-key attributes as the parent has primary-key attributes. Figure 32 on page 215 provides an example.

In each case, the foreign-key relationship between the parent and child business objects is specified by the application-specific information of the key attributes of the child business object. For more information, see "Business object attribute properties" on page 219 and "Application-Specific information for simple attributes" on page 221. For information on how Advanced Outbound Wizard handles these two cases, see "How does the wizard create relationships between tables?" on page 226.

Single-cardinality relationship example

Figure 30 on page 214 provides an example of a simple WebSphere business object developed to process customer objects in SAP. This example SAP_Customer has a single-cardinality relationship to the example address object that it contains (the `addr_data[1]` attribute has cardinality 1). The primary key attribute (`address_id`) in the child business object references a non-primary key (`address_id`) in the parent business object.

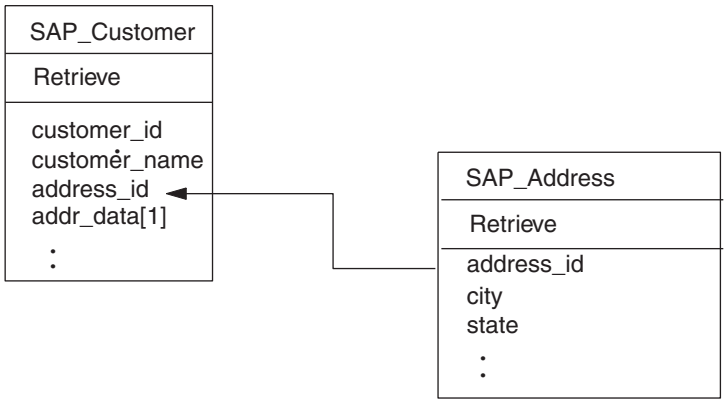


Figure 30. Example customer and address relationship

The following SELECT statements and their output illustrate retrieval of data from the tables represented by the business objects above:

```
SELECT * FROM KNA1

KUNNR          NAME1          ADRNR
-----          -------------  ----------
10254          JOE'S PIZZA          2208
10255          LARRY'S HARDWARE     2209

SELECT * FROM ADRC

ADDRNUMBER          CITY1          REGION
----------          ----           -----
2208                BURLINGAME     CA
2209                SAN FRANCISCO  CA
```

In the example above, each customer (Joe's Pizza and Larry's Hardware) has a single address. If the KUNNR and ADDRNUMBER columns are defined as primary key constraints for their respective tables, the above structure ensures that each customer can have only one associated address.

Note: For the sake of simplicity, the illustrations in this document do not display the application-specific information used by the connector to determine the tables and fields in the SAP application's database.

Multiple-cardinality relationship example

Figure 31 on page 215 illustrates a multiple-cardinality relationship. In the example, `ID=ABC` is the simple attribute with the parent's primary key, and `child[n]` is the attribute that represents the array of child business objects.

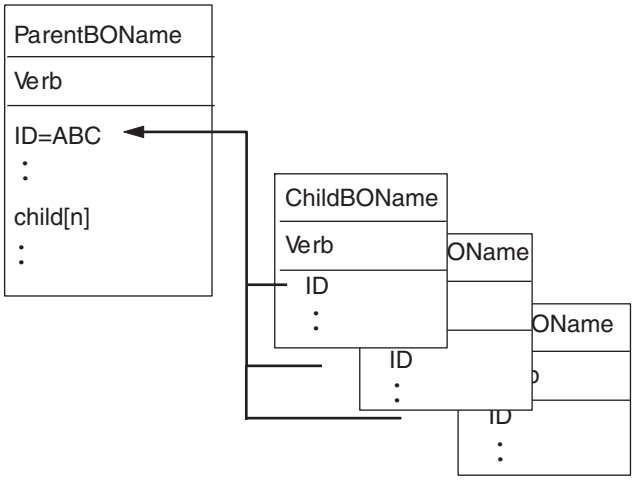


Figure 31. Multiple-cardinality business object relationship

Figure 32 provides an example of a different WebSphere business object developed to process customer objects in SAP. This example SAP_Customer has a multiple-cardinality relationship to the example sales view object that it contains (the `sales_view_data[n]` attribute has cardinality `n`). The primary key attribute (`customer_id`) in the child business object references the primary key (`customer_id`) in the parent business object.

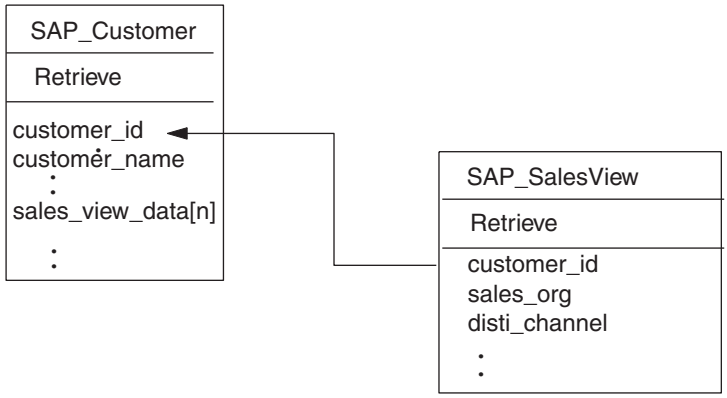


Figure 32. Example customer and sales view relationship

The following `SELECT` statements and their output illustrate retrieval of data from each of these tables:

```
SELECT * FROM KNA1

KUNNR          NAME1
-----          -------------
10254          JOE'S PIZZA
10255          LARRY'S HARDWARE

SELECT * FROM KNVV

KUNNR          VKORG          VTWEG          SPART
----------     -----          -----          -----
10254          EURP          01             12
10255          EURP          01             09
10255          USA           01             13
10255          USA           01             14
```

In this example, Joe's Pizza has one associated sales view record, whereas Larry's Hardware has three associated sales view records. The above structure allows each customer to have zero or more associated sales view records.

Handling long data rows

SAP's RFC_READ_TABLE function limits data retrieval to 512 bytes per row of data. Many SAP tables have more than 512 bytes of data per row. However, most business objects represent a small subset of all the database fields. Therefore, the total length of all attributes in a business object rarely exceeds the 512 byte maximum.

In those cases that require the connector to retrieve more than 512 bytes of data from a single database table, the additional fields must be represented in separate single-cardinality child business objects. For example, if a business object must represent 1500 bytes of data from a single table, the top-level business object contains at least two single-cardinality child business objects. Neither the parent nor either child has attributes whose total length (that is, the sum of their maximum length) exceeds 512 bytes.

Note: If a business object represents more than one database table, the total length of the values in the attributes that represent each table cannot exceed 512 bytes. However, this limit does not pertain to the total length of the values of all attributes. For example, if a business object represents data from the tables that store information about Customers and CustomerPartners, the value of those attributes representing Customers cannot exceed 512 bytes, and the value of those attributes representing CustomerPartners cannot exceed 512 bytes, but the combined value of these attributes can exceed 512 bytes.

Important: When you use Advanced Outbound Wizard to create business object definitions, and it encounters an object that represents more than 512 bytes of data from a single table, it stops adding attributes to the child business object when the length exceeds 512 bytes. If your business processing requires a business object to represent more than 512 bytes of data from a single table, you must manually create the additional child business objects.

Business object verb processing

This section outlines the steps the connector takes to handle a business object request with the Retrieve verb. 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

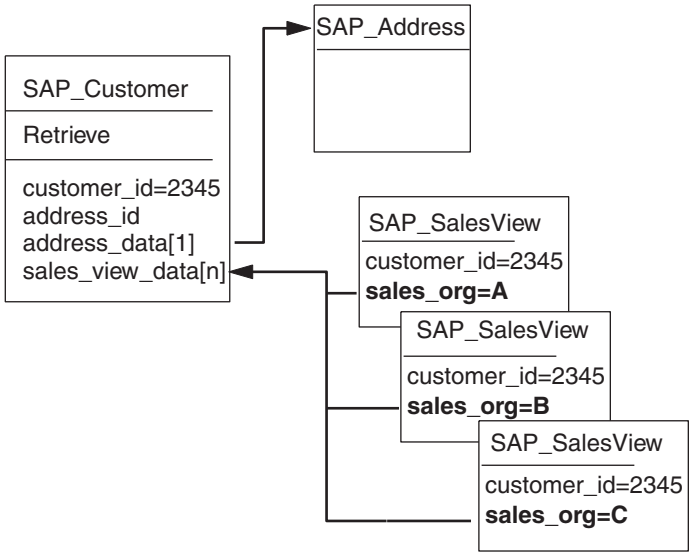
When processing a retrieval request from the integration broker, the connector tries to return a business object that matches the current database representation of that object. In other words:

- The value of each simple attribute in all individual business objects returned to the integration broker matches the value of its corresponding field in the database.
- The number of individual business objects in each array of the returned business object matches the corresponding number of children in the database.

Therefore, when the Hierarchical Dynamic Retrieve Module receives a business object request with the Retrieve verb, it creates a response business object by

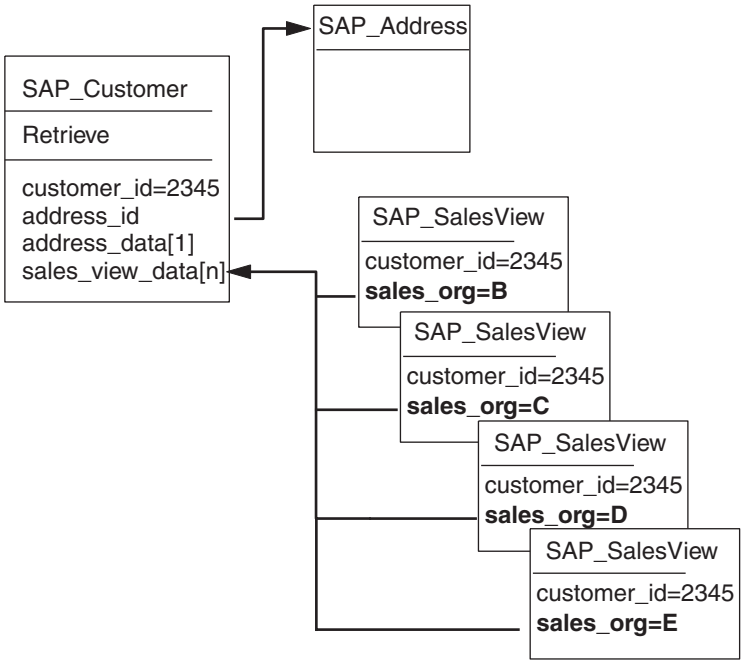
recursively descending the entire object in the application and retrieving the current database representation. To perform the retrieval, the connector uses the specified key values in the top-level request business object. Therefore, the response business object, which contains all the children of that top-level parent, may have different values for simple attributes and different child business objects from the request business object.

For example, assume the integration broker passed the following SAP_Customer business object to the Hierarchical Dynamic Retrieve Module:



If, in the current database representation, the array of SAP_SalesView child business objects contained by SAP_Customer 2345 does not include sales_org A, the connector's response business object does not contain that child. Moreover, if the current database representation of SAP_Customer 2345 includes sales_org D and sales_org E, the connector includes those children in the response business object. The business object that the SAP Hierarchical Dynamic Retrieve Module

returns to the integration broker at the end of retrieval is:



Note: If the connector reads from multiple tables when creating a particular response business object, the business object does not match a single database object. Instead, it matches selected fields from the specified tables.

Retrieve Operation

When retrieving 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 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. Calls the RFC_READ_TABLE function to retrieve the **top-level business object** from the database.

   The connector uses key values in the request business object to build the `SELECT` statement's `WHERE` clause. The result of the retrieval causes one of the following actions:

   - If the `SELECT` statement returns one record, the connector continues processing the children and returns `VALCHANGE` (regardless of whether any attribute changed value).
   - If the `SELECT` statement returns no records, indicating that the top-level business object does not exist in the database, the connector returns `BO_DOES_NOT_EXIST`.
   - If the `SELECT` statement returns more than one record, the connector continues processing the children and returns `VALCHANGE`.
3. Recursively retrieves all **child business objects** (single-cardinality and multiple-cardinality).

The connector calls the RFC_READ_TABLE function, which uses the appropriate foreign-key values to build the SELECT statement's WHERE clause. The connector handles attributes marked as required in the following way:

- If the business object's definition specifies that the child is required, the retrieval must return a record. If not, the connector returns FAIL.
- If the child is not required and the retrieval returns no records, indicating that the child does not exist in the application, the connector leaves the parent's attribute empty.

For each record 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.

   Attention: If the retrieval of a single-cardinality child returns more than one record, the connector returns only the first record.
d. Inserts the current business object with all of its children into the appropriate single-cardinality attribute or array attribute of the parent.

Note: A business object can have attributes that do not correspond to any database column, such as placeholder attributes. During retrieval, the connector does not change such attributes in the top-level business object; they remain set to the values received from the integration broker. The application-specific information for these attributes must be blank.

Business object attribute properties

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

Name property

Each business object attribute must have a unique name.

Type property

Each business object attribute must be of type String, or the type of a child business object or an array of child business objects.

Cardinality property

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

Max length property

The connector does not use this property. Although Advanced Outbound Wizard populates this property when it generates the business object, it does so only to provide information.

Key property

At least one simple attribute in each business object must be specified as the key. To define an attribute as a key, set this property to true.

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

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

To maximize performance, it is recommended that you provide data for as many key fields as possible.

To retrieve a child business object or children from an array of business objects, the connector uses foreign keys in the `WHERE` clause of the `SELECT` statement. It does not use the Key property of attributes in child business objects. For information on how to specify an attribute in a child business object as a foreign key, see "Application-Specific information for simple attributes" on page 221.

Foreign key property

The connector does not use this property. The connector obtains foreign-key information from application-specific information. For more information, see "Application-Specific information for simple attributes" on page 221.

Required property

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

- If an attribute that represents a child business object or an array of child business objects is marked as required and the connector fails to retrieve any child from the application, the retrieve operation fails.
- If a simple attribute is marked as required and the connector fails to retrieve the corresponding row from the database, the retrieve operation fails. For example, if the connector reads from multiple tables for a business object and it fails to retrieve a row for a required simple attribute that represents a value in one of the tables, the entire retrieve fails.

AppSpecificInfo

For information on this property, see "Application-Specific information for simple attributes" on page 221.

Default value property

This property specifies a default value that the connector uses when generating the `WHERE` clause of a `SELECT` statement. This property is relevant only to simple attributes that have been specified as key. For example, to cause the connector to use the default value specified for the Language attribute, you must specify the Language attribute as key.

Special value for simple attributes

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

When the connector retrieves data from the database and the `SELECT` statement returns a blank value for an attribute, the connector sets the value of that attribute to `CxBlank` by default.

Because the connector requires every business object to have at least one key attribute, make sure that business objects passed to the connector have at least one primary or foreign key that is not set to `CxIgnore`.

Business object application-specific information

Application-specific information in business object definitions provides the connector with application-dependent instructions on how to process business objects. This information includes:

- The class for the vDynRetBOH business object handler, which is provided in the application-specific information for the verb of the top-level business object. This value is identical for all business objects that this module processes.
- Database and query information, which is provided in the application-specific information for simple attributes. The connector parses this information to generate `SELECT` queries.

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

The following sections discuss this functionality in more detail.

Application-specific information for the top-level business object's verb

The verb of the top-level business object specifies the class for the vDynRetBOH business object handler. This application-specific information should always be the following:

```
sap.bapimodule.vDynRetBOH
```

Application-Specific information for simple attributes

The application-specific information for attributes specifies the following information:

- The name of the corresponding database table
- The name of the corresponding database column
- The foreign key relationship between an attribute in the current business object and a parent or child business object
- The operand

The application-specific information format consists of four name-value parameters, each of which includes the parameter name and its value. Each parameter set is delimited from the next by a colon (:).

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

```
TN=TableName:CN=ColumnName:[FK=[..]fk_attributeName]:[OP=GT|GE|EQ|NE|LE|LT|LIKE]
```

Table 46 describes each name-value parameter.

Table 46. Name-value parameters in attribute application-specific information

Parameter	Description
`TN=TableName`	The name of the database table

Table 46. Name-value parameters in attribute application-specific information (continued)

Parameter	Description
CN=*ColumnName*	The name of the database table column (field)
FK=[..]fk_*attribute Name*	The value of this property depends on whether the foreign-key relationship is stored in the parent business object or the current business object: • *attributeName*—specifies an attribute in the current business object; for more information, see "Example: Current business object stores the foreign key" • ..attributeName—specifies an attribute in the parent business object If an attribute is not a foreign key, do not include this parameter in the application-specific information.
OP=GT|GE|EQ|NE|LE|LT| LIKE	The operand options are: • GT—Greater Than • GE—Greater than or Equal to • EQ—EQual to (default option) • NE—Not Equal to • LE—Less than or Equal to • LT—Less Than • LIKE—Like It is recommended that you specify EQ to maximize performance. If no operand is specified, the connector uses EQ.

The required parameters for each simple attribute are the table name and column name. The operand defaults to EQ (equals). The following example illustrates the basic format:

```
TN=KNA1:CN=KUNNR
```

Important: Case is significant when specifying values for these parameters.

It is permissible for simple attributes within a business object to have no value specified (that is, zero length) for application-specific information fields. The connector ignores such attributes. This is a convenient way to ensure that the connector does not process placeholder attributes used to separate adjacent arrays of child business objects.

If none of the application-specific information in any of a business object's attributes provide sufficient information for the connector to build or execute a query, the connector returns a failure.

Example: Current business object stores the foreign key

Figure 33 on page 223 provides an example of a WebSphere business object with two foreign keys that reference attributes within the business object. In this case, the business object represents data in two tables, one containing address data and the other containing lookup data for state/province and country abbreviations. To process this data, the connector performs two table reads.

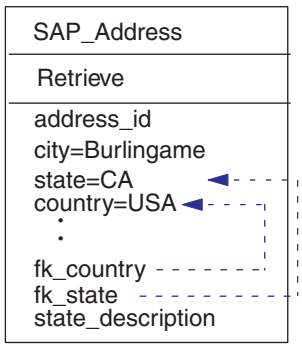


Figure 33. Example: current business object stores the foreign key

Attribute information: Table 47 documents the table name, column name, key, and foreign-key for each attribute in the example SAP_Address:

Table 47. Description of example business object attributes

Attribute	Table name	Column name	Key	Foreign key	Default
address_id	ADRC	ADDRNUMBER	true		
city	ADRC	CITY1	false		
state	ADRC	REGION	false		
country	ADRC	LAND1	false		
language	T005U	SPRAS	true		E
fk_country	T005U	LAND1	false	FK=country	
fk_state	T005U	BLAND	false	FK=state	
state_description	T005U	BEZEI	false		

Attribute application-specific information: Given the information in Table 47, the application-specific information for the fk_state attribute is:

```
TN=T005U:CN=BLAND:FK=state
```

The application-specific information for the fk_country attribute is:

```
TN=T005U:CN=LAND1:FK=country
```

SQL Queries: The following `SELECT` statements illustrate the `WHERE` clause that the connector builds to retrieve data from the tables represented by SAP_Address:

```
SELECT * FROM ADRC WHERE ADDRNUMBER = address_idValue
SELECT * FROM T005U WHERE SPRAS = 'E' AND LAND1 = countryValue AND
BLAND = stateValue
```

Generating business objects

The WebSphere business integration system provides two utilities that enable you to define business objects and the metadata required to support the processing of those business objects in the SAP application:

- SAPODA generates business object definitions from tables you specify graphically. This utility is most useful for creating individual business object definitions rather than hierarchical business object definitions. You must manually define the relationships between parent and child business objects.
- Advanced Outbound Wizard records and interprets your actions as you step through an SAP Display Transaction. As it generates the business object definition, it automatically defines the relationships between parent and child business objects.

Generating business objects: SAPODA

SAPODA generates individual business object definitions for the Hierarchical Dynamic Retrieve Module. If you use this utility to create hierarchical business object definitions, you must manually specify the relationships between the generated parent and child business object definitions.

Note: Table definition anomalies may produce definitions that require manual changes to fully meet your needs.

Steps to creating a business object definition with SAPODA

To use SAPODA to generate a business object definition for this module:

1. Launch SAPODA.
2. Launch Business Object Designer, which is the utility that facilitates development of business object definitions, both manually and automatically (by providing access to ODAs).
3. Follow a six-step process in Business Object Designer to configure and run the ODA.
4. Use Business Object Designer to manually modify the generated definition:
   - Remove unwanted attributes.

     Important: Because the total number of bytes for all columns in each table represented by a single a business object cannot exceed 512, you must remove unnecessary attributes whose length causes the definition to exceed this limit. For more information, see "Handling long data rows" on page 216.
   - If creating a hierarchical business object definition, specify the relationships between parent and child business objects.
   - Remove undesirable anomalies.

For information on using SAPODA, see Appendix E, "Generating business object definitions using SAPODA," on page 291. For information on launching Business Object Designer and using it to manually modify a business object definition, see the *Business Object Development Guide*.

Creating relationships between tables

SAPODA generates a business object definition for every table you specify. When it completes generating, you can open all tables in Business Object Designer for editing.

To create a hierarchical business object definition from the individual business object definitions generated by SAPODA, do the following:

1. Determine the table at the top of the hierarchy.

   Assume, for example, the top-level business object is SAP_Customer. This business object has a single key, Customer_KUNNR. SAPODA specifies the following application-specific information for this attribute:

   ```
   TN=KNA1:CN=KUNNR
   ```
2. Locate and differentiate every child and grandchild business object.
3. To the top-level business object and to each parent in the hierarchy below it, add an attribute that represents each child business object or array of child business objects:
   - Specify the name of the child as the type of the attribute.
   - Specify containment as the relationship.

- Specify the appropriate cardinality, either 1 or n.

4. To each child business object definition that contains an attribute corresponding to the key of its parent, specify the foreign-key relationship in the attribute's application-specific information.

   For example, most business objects that are a direct child of SAP_Customer contain the Customer_KUNNR attribute. In the application-specific information for Customer_KUNNR, specify the following:

   ```
   TN=KNVI:CN=KUNNR:FK=..Customer_KUNNR
   ```

   For information on specifying foreign keys, see Table 46 on page 221.

5. Locate child business object definitions whose corresponding tables do not contain the key of the parent object. In these definitions, locate a non-key field in the parent that matches the child's primary key.

   For example, SAP_Customer_ADRC is a second-level business object with no key the same as its parent's. SAPODA generates this business object definition with the Address_number_ADDRNUMBER attribute, which is a non-key field in SAP_Customer.

   In the application-specific information for this attribute, specify the foreign-key relationship as:

   ```
   TN=ADRC:CN=ADDRNUMBER:FK=..Address_ADRNR
   ```

   Note: Because SAP changed the name of the ADDNR field used in tables (such as KNA1) created in SAP Version 3x to ADDRNUMBER in tables (such as ADRC) created in SAP Version 4x, recognizing the relationship between these two fields is more challenging.

Generating business objects: Advanced Outbound Wizard

The WebSphere business integration system enables users to define business objects and the metadata required to support the processing of those business objects in the SAP application. Advanced Outbound Wizard records and interprets your actions as you step through an SAP Display Transaction.

Important: The wizard is intended to assist business object development. Due to table definition anomalies, the business object produced may require manual changes to fully meet your needs.

The wizard supports the definition of flat and hierarchical business objects that use the Retrieve verb.

Note: Before you use the wizard, ensure that your existing entity in SAP includes all the information required for the business object. For example, for a sales order, ensure that the entity you use has the line items, schedule lines, and partners you require.

Steps to Creating a business object definition with advanced outbound wizard

To use Advanced Outbound Wizard to generate a business object definition for this module:

1. Go to IBM CrossWorlds Station (transaction /N/CWLD/HOME).

   Important: You must log on to the SAP system in English when using IBM CrossWorlds Station to generate business object definitions. The CrossWorlds Station log is available only in English.

2. On the Development tab, click the Advanced Outbound Wizard Button.

3. In the IBM CrossWorlds outbound wizard window, enter the following information:
   - IBM CrossWorlds business object name—Name of the business object type as well as the name of every instance of the object. It is recommended that you use a simple name that identifies the business object; for example, enter `SAP_Customer`.
   - Transaction Code—Transaction code for the transaction that supports the necessary functionality performed by the business object; for example, specify `XD03`, the transaction that displays a customer centrally.
4. Click Record.
5. In the initial display screen, select the information you want the transaction to process.
6. Step through the transaction. When you reach the last screen, you will be asked whether to exit the display. Click the Yes button.

   The CW Hierarchical Outbound Wizard screen displays. The business object displays closed at the top of the screen.
7. Click the plus button (+) to the left of the business object's name to view the business object's attributes. Scroll to the bottom of the displayed attributes and click the plus button (+) to the left of the names of its child business objects to view their attributes.

   The wizard displays each field's length to the right of its name. To the right of the field's length, the wizard displays cumulative field length. If the cumulative length exceeds 512 bytes, the wizard marks the field with a red X and turns the field's display from yellow to red. By default, the wizard stops adding attributes to a business object when the length exceeds 512 bytes. To include desired attributes that are removed by default, you can:
   - Remove unwanted attributes, as described in the next step.
   - Manually modify the business object definition to contain additional child business objects, as described in "Handling long data rows" on page 216.
8. Remove unnecessary attributes from the definition by double-clicking their row. The color of deleted rows turns blue.
9. Verify that the business object definition includes all desired attributes. When you are satisfied with the definition, click the generate button.

   The Download Object Definition screen displays.
10. On the Download Object Definition screen, specify the location and name of the business object definition file and click Save.

For information on the business object and attribute names created by the wizard, see "Business object names" on page 212.

How does the wizard create relationships between tables?

When generating the business object definition, Advanced Outbound Wizard creates a list of all tables involved in the SAP transaction. You can view the list of tables by pressing the Show Tables button at the top of the wizard's screen. Table 48 illustrates the tables used to generate SAP_Customer. Table.

Table 48. illustrates the tables used to generate SAP_Customer.

Name	Table Description
TFDIR	Function module
KNA1	General Data in Customer Master
T001	Company Codes

Table 48. illustrates the tables used to generate SAP_Customer. (continued)

Name	Table Description
KNB1	Customer Master (Company Code)
KNVV	Customer Master Sales Data
TOBJ_OFF	Objects that were disabled
ADRC	Addresses (central address admin.)
ADRCT	Address texts (central address admin.)
ADRG	Assignment of addresses to other address groups (cent.adr.)
ADRV	Address where used list (central address administration)
T002	Language keys
TBE01	Business Transaction Events: Publish & Subscribe Interfaces
TBE31	Application components per Publish & Subscribe interface
TBE32	Partner function modules per Publish & Subscribe interface
TBE34	Customer function modules per Publish & Subscribe interface
T100	Messages
DOKIL	Index for Documentation Table DOKH
KNVI	Customer Master Tax Indicator
TVKWZ	Org.Unit: Allowed Plants per Sales Organization
T001W	Plants/branches
KNVL	Customer Master Licenses
TSADVC	Customizing international address versions
TMODU	Cross Reference Field Name - MODIF1
TCONV_ADR	Conversion of Old Address Fields to CAM Address Fields
TSAD7	Address groups (central address management)
T005T	Country Names
T005U	Taxes: Region Key: Texts
TZONT	Customers: Regional Zone Texts
TSAD7T	Description of address groups (central address admin.)
TOJTD	Customizing Object Types
TOJTB	Business object repository: Basic data

The wizard makes three passes through the tables to determine their hierarchy and the relationship among them. It uses the information to specify foreign-key relationships in the application-specific information of business object attributes. It names the attributes based on the field's description in SAP's data dictionary in the user's language.

In its three passes, the wizard does the following:

1. Determines the table at the top of the hierarchy.

   Typically this is the first table the wizard locates that has only one key field.
2. Determines child business objects that contain the key of the top-level business object, and differentiates child from grandchild relationships based on the number of key fields.

Typically several tables contain the parent's key field as well as multiple other keys. Grandchild tables typically contain more keys than their parent. Tables with identical keys are siblings (peers) of the parent.

3. Determines the relationship of tables that do not contain the key of the top-level business object.

   The wizard establishes the relationship by locating a non-key field in the parent that matches the child's primary key.

Table 48 on page 226 illustrates the results of all three of these passes through the list of display-Customer tables:

- Top-level business object: SAP_Customer

  It has a single key, Customer_KUNNR. The wizard specified the following application-specific information for this attribute:

  ```
  TN=KNA1:CN=KUNNR
  ```

- Second-level business object that contains its parent's key: SAP_Customer_KNVI

  The wizard identified the following three key fields, one of which is found as a key field in the parent:

  – Customer_KUNNR
  – Country_ALAND
  – Tax_category_TATYP

  In the application-specific information for the Customer_KUNNR attribute, the wizard specified the foreign-key relationship to the key field in the parent:

  ```
  TN=KNVI:CN=KUNNR:FK=..Customer_KUNNR
  ```

  For information on specifying foreign keys, see Table 46 on page 221.

- Second-level business object with no key the same as its parent's: SAP_Customer_ADRC

  The wizard identified the following three key fields, one of which is found as a non-key field in the parent:

  – Address_number_ADDRNUMBER
  – from_DATE_FROM
  – Address_version_NATION

  In the application-specific information for the Address_number_ADDRNUMBER attribute, the wizard specified the foreign-key relationship to a non-key field in the parent:

  ```
  TN=ADRC:CN=ADDRNUMBER:FK=..Address_ADRNR
  ```

  Note: Because SAP changed the name of the ADDNR field used in tables (such as KNA1) created in SAP Version 3x to ADDRNUMBER in tables (such as ADRC) created in SAP Version 4x, recognizing the relationship between these two fields is more challenging. To handle the challenge, the wizard is coded to recognize the link and create the appropriate foreign-key relationship.

Attention: Verify the definition that the wizard generates. Although the wizard performs most of the work in creating a business object definition, the generated definition cannot meet the precise needs of your implementation. There are anomalies in table relationships that may cause the wizard to make incorrect decisions. For example, if two tables have the same set of keys and a child table has the same keys plus one, the wizard assigns the child to the first table.

Part 7. Appendixes

Appendix A. Quick Steps

This appendix supplements information contained in the *Adapter for mySAP.com User Guide*. It is not intended to replace the user guide.

Before you begin these steps, you must:

- Install WMQI as your broker.
- Install the SAP Java Connector (SAPjco). Download SAPjco for your connector's operating system from: http://service.sap.com/connectors. Add these files to the `%CROSSWORLDS%\ODA\SAP` and `%CROSSWORLDS%\connectors\SAP` directories.
- Install the JDK.
- Install CrossWorlds.
- Configure standard MQ Queues

Common configuration properties

The following tables list configuration properties that must be maintained for the WMQI broker. Create the SAP configuration file using `CN_SAP.txt`. This file is located in `%CROSSWORLDS%\repository\SAP`. Open the file using Connector Configurator.

Table 49. Standard configuration properties

Property name	Default Value	Value Needed
ApplicationName	none	SAPConnector
BrokerType	ICS	WMQI
AdminInQueue	/ADMININQUEUE	ADMININQUEUE
AdminOutQueue	/ADMINOUTQUEUE	ADMINOUTQUEUE
DeliveryQueue	/DELIVERYQUEUE	DELIVERYQUEUE
FaultQueue	/FAULTQUEUE	FAULTQUEUE
RequestQueue	/REQUESTQUEUE	REQUESTQUEUE
ResponseQueue	/RESPONSEQUEUE	RESPONSEQUEUE
SynchronousRequestQueue	/SYNCHRONOUSREQUESTQUEUE	SYNCHRONOUSREQUESTQUEUE
SynchronousResponseQueue	/SYNCHRONOUSRESPONSEQUEUE	SYNCHRONOUSRESPONSEQUEUE
MessageFileName	SAPConnector.txt	SAPCONNECTOR.TXT
RepositoryDirectory	C:\crossworlds\repository	*<location of business object specifications>*
Jms.MessageBrokerName	crossworlds.queue.manager	*<Queue Manager name>*
AgentTraceLevel	0	5

Table 50. Connector-specific properties

Property Name	Default Value	Value Needed
ApplicationPassword	SOFTWARE	*<password for SAP application>*

Table 50. Connector-specific properties (continued)

Property Name	Default Value	Value Needed
ApplicationUserName	CROSSWORLDS	*<username for SAP application>*
Client	none	*<Client number>*
Hostname	none	*<SAP application server name>*

Quick steps for the BAPI module

Before configuring the BAPI module, configure the following connector-specific property:

Property Name	Default Value	Value Needed
Modules	Extension	BAPI

Generating a business object in the BAPI module

To generate a business object for the BAPI module:

1. Start the SAP ODA.
2. Start the business object designer.
3. In the business object designer, choose File > New.
4. Select the Configure Discovery:
   a. Enter the host address for the machine where Discovery is running.
   b. Choose Add Host.
   c. Choose OK.
5. Choose Find Agents.
   a. Highlight Agent. Choose Next.
   b. Populate the values for UserName, Password, Client, SystemNumber, ASHostName, and FileDestination. Save the profile.
6. In Step 3 of the wizard, expand the RFC node.
   a. Right-click Search By Name.
   b. Type `bapi_customer_getdetail`.
   c. Highlight `bapi_customer_getdetail`.
   d. Choose Next.
7. Choose Next.
8. Set Verb to **Retrieve**, and Server Support to **No**. Choose OK.
9. In Agent SAPODA Notification, choose No.
10. Open the business object in a separate window. Save the generated business object specification to the location you specified in the Repository Directory standard property value.

Configuring the BAPI module

After you have generated a business object, continue configuring the BAPI module:

1. Add the parent object name to the Supported Business Object section of the configuration file.

2. Copy the generated BOHandler .class file from the file definition specified in the ODA configuration properties to `%CROSSWORLD%\connectors\SAP\bapi\client`.

Preparing the BAPI module for testing

To set up the BAPI module for testing, use Port Connector:

1. Copy the SAP configuration file. Rename the copied file `portconnector.cfg`.
2. Open portconnector.cfg in Connector Configurator.
3. Change the following properties in the Standard tab:
   - `ApplicationName` to `PortConnector`
   - `DELIVERYQUEUE` to `REQUESTQUEUE`
   - `REQUESTQUEUE` to `RESPONSEQUEUE`
4. Save changes. Close `portconnector.cfg`.
5. Open `sapconnector.cfg`.
6. Save the change. Start mySAP.com.

Testing the BAPI module

To test the BAPI module:

1. Open Test Connector.
2. Choose File > Create/Select Profile.
3. Choose File > New Profile.
4. Select Browse.
   a. Locate `portconnector.cfg`. Choose Open.
   b. For Connector Name, enter `PortConnector`.
   c. For Broker Type, enter `WMQI`.
   d. Choose OK.
5. Highlight `PortConnector`. Choose OK.
6. Choose File > Connect.
7. Create a Business Object Instance:
   a. For BO Type select `SAP_BAPI_customer_getdetail`.
   b. Choose Create.
   c. Enter New Object. Choose OK.
8. Change the Verb to Retrieve. Populate `Customer_to_be_required` with an existing customer.
9. Choose Request > Send.
10. Check the log file for a success message.

Quick steps for the RFC Server module

Before configuring the RFC module, configure the following connector-specific properties:

Property Name	Default Value	Value Needed
`Modules`	`Extension`	`Rfcserver`
`RfcProgramId`	`CWLDSERVER`	*<ProgramId registered in SAP transaction sm59>*

Generating a business object in the RFC Server module

To generate a business object for the RFC module:

1. Start the SAP ODA.
2. Start the business object designer.
3. In the business object designer, choose File > New.
4. Select the Configure Discovery:
   a. Enter the host address for the machine where Discovery is running.
   b. Choose Add Host.
   c. Choose OK.
5. In Step 3 of the wizard, expand the RFC node.
   a. Right-click Search By Name.
   b. Type `bapi_customer_getdetail`.
   c. Highlight `bapi_customer_getdetail`.
   d. Choose Next.
6. Choose Next.
7. Set the Verb to Retrieve and Server Support to No. Choose OK.
8. In Agent SAPODA Notification, choose No.
9. Open the business object in a separate window. Choose General > Set Collab = ″RFCCollab″.
10. Save the generated business object specification to the location you specified in the Repository Directory standard property value.

Configuring the RFC Server module

After you have generated a business object, continue configuring the RFC Server module:

1. Add the parent object name to the Supported Business Object section of the configuration file.
2. Copy the generated BOHandler .class file from the definition specified in the ODA configuration properties to `%CROSSWORLD%\connectors\SAP\rfc\client`.

Creating a profile for the SAP server

To create a profile for the SAP server:

1. Open SAP Logon.
2. Choose New.
3. Populate the following fields, then choose OK:

Description	Hostname of server
Application server	Hostname of server
System Number	00
Description	Hostname is standard. Enter a description of your choice.

4. Double-click to open the profile you just created.
5. Enter your username and password. Choose Transaction > Type /nse37. Function Builder opens.
6. For Function Module, input `bapi_customer_getdetail`. Choose Funtion Module >Test > Single Test.

7. For the RFC target system, use the value for `Rfcprogramid` you set in the connector-specific properties. Also populate the following fields:

Field	Example
`Customer Number`	0000000001
`PI_SALESORG`	0001
`PI_DISTR_CHAN`	01
`PI_DIVISION`	01

Testing the RFC server module

To set up the BAPI module for testing, use Port Connector:

1. Copy the SAP configuration file. Rename the copied file `portconnector.cfg`.
2. Open `portconnector.cfg` in Connector Configurator.
3. Change the following properties in the Standard tab:
   - `ApplicationName` to `PortConnector`
   - `REQUESTQUEUE` to `SYNCHRONOUSREQUESTQUEUE`.

   Save the changes and close the window.
4. Open `sapconnector.cfg`.
5. Change `REQUESTQUEUE` to `SYNCHRONOUSREQUESTQUEUE`. Save the change.
6. Start the connector. Choose Function Module > Execute.
7. In Test Connector, find the object in BO Request List. Highlight the object, and choose Request > Reply > Success.
8. Check the log for a success message.

Quick steps for the ALE module

Before you configure the ALE module, create the following persistent WebSphere MQ queues:

- `SAPtid_Queue`
- `SAPtid_QueueManager`
- `SAPALE_Event_Queue`
- `SAPALE_Wip_Queue`
- `SAPALE_Archive_Queue`
- `SAPALE_UnSubscribed_Queue`
- `SAPALE_Error_Queue`

Refer to the MQ Series documentation for information on creating MQ Queues.

Next, configure the following connector-specific properties:

Property Name	Default Value	Value Needed
`Modules`	Extension	`Ale`
`AleEventDir`	none	`%CROSSWORLDS%\connectors\SAP\ale`
`SAPtid_QueueManager`	none	*<Queue Manager name>*
`SAPtid_Queue`	none	*<Queue name>*
`SAPALE_Event_Queue`	none	*<Event Queue name>*
`SAPALE_Wip_Queue`	none	*<WIP Queue name>*

Property Name	Default Value	Value Needed
SAPALE_Archive_Queue	none	*<Archive Queue name>*
SAPALE_UnSubscribed_Queue	none	*<UnSubscribed Queue name>*
SAPALE_Error_Queue	none	*<Error Queue name>*
RfcProgramId	none	*<Program ID name defined in SAP Transaction sm59>*
NumberOfListeners	1	1 (for single-threaded)

For remote WebSphere Queues, also configure the following properties:

Property Name	Default Value	Value Needed
SAPtid_QueueManagerLogin	none	*<Queue Manager login>*
SAPtid_QueueManagerPassword	none	*<Queue Manager password>*
SAPtid_QueueManagerHost	none	*<Queue Manager host>*
SAPtid_MQPort	none	*<MQ port>*
SAPtid_MQChannel	none	*<MQ channel>*

Generating a business object in the ALE module

To generate a business object in the ALE module:

1. Start the SAP ODA.
2. Start the business object designer.
3. In the business object designer, choose File > New.
4. Select the Configure Discovery:
   a. Enter the host address for the machine where Discovery is running.
   b. Choose Add Host.
   c. Choose OK.
5. In Step 3 of the wizard, expand IDoc Types.
   a. Expand Generate From System.
   b. Expand Basic IDoc Types.
   c. Right-click Select by Name...
   d. Select Search for Items...
   e. Type `orders03`. Choose OK.
6. Highlight `ORDERS03`. Choose Next.
7. Choose Next.
8. Choose OK. The business object generates.
9. Select ″Save a copy of business object definitions to a separate file″ and select ″Open new business object definition to a separate window″. Choose Finish.

Editing the business object.

To edit the business object:

1. Choose the General tab.
2. Change Create Application-specific information message type to `MsgType = ORDERS`.
3. Open `%CROSSWORLDS\repository\SAP\BO_SAPIDocControl.txt` and save it to the Repository directory.

4. Add the parent object name to the Supported Business Objects section of the configuration file.
5. Register the RFC Server Module with the SAP Gateway, using SAP transaction SM59.
6. Ensure the following:
   - That the logical systems are defined and assigned for the SAP system and external system (SALE).
   - That the distribution model has been maintained and that the required message types have been added to the model (transaction code BD64).
   - That there are partner profiles for the logical systems or distribution model (transaction code WE20).
   - That there are ports defined for the logical systems or distribution model (transaction code WE21).

Preparing the ALE module for testing

To set up the ALE module for testing, use Port Connector:

1. Copy the SAP configuration file. Rename the copied file `portconnector.cfg`.
2. Open portconnector.cfg in Connector Configurator.
3. Change the following properties in the Standard tab:
   - `ApplicationName` to `PortConnector`
   - `DELIVERYQUEUE` to `REQUESTQUEUE`
   - `REQUESTQUEUE` to `RESPONSEQUEUE`
4. Save changes. Close `portconnector.cfg`.
5. Open `sapconnector.cfg`.
6. Save the change. Start mySAP.com.

Testing request processing for the ALE module

To test the ALE module:

1. Open Test Connector.
2. Choose File > Create/Select Profile.
3. Choose File > New Profile.
4. Select Browse.
   a. Choose Open.
   b. For Connector Name, enter `PortConnector`.
   c. For Broker Type, enter `WMQI`.
   d. Choose OK.
5. Highlight `PortConnector`. Choose OK.
6. Choose File > Connect.
7. Create a business object instance:
   a. For BO Type, select `sap_order03`.
   b. Choose Create.
   c. In Enter Name, type new object. Choose OK.
8. Change the verb to Create.
9. Right-click Control Record. Choose Add Instance.
10. Expand Control Record. Populate these fields:
    - `IDoc_number`

- Sender_port
- Partner_number_of_sender
- Receiver_port
- Partner_number_of_recipient
- Client
- SAP_Release

11. Start the connector.
12. In Test Connector, choose Request > Send. Check the log for a success message.

Testing event processing in the ALE module

To test event processing in the ALE module:

1. Go to transaction we19, Test Tool for IDoc processing.
2. Populate the field with an existing IDoc. Choose IDoc > Create.
3. Choose StandardOutboundProcessing to send an IDoc to the test connector.
4. In the pop-up window, choose the check mark.
5. To verify that the IDoc was sent from SAP, check the mySAP.com connector log file for a success message. If the event exists in transaction sm58, then it was not sent correctly.
6. View the message that was sent to the SAPALE_Archive_Queue to verify that the ProcessingStatus was successful. If you do not see a success message, check the SAPALE_Error_Queue to see if a failure occurred.

Quick steps for the HDR module

Before configuring the HDR module, configure the following connector-specific property:

Property Name	Default Value	Value Needed
Modules	Extension	BAPI

Generating a business object in the HDR module

To generate a business object in the HDR module:

1. Start the SAP ODA.
2. Start the business object designer.
3. In the business object designer, choose File > New.
4. Select the Configure Discovery:
   a. Enter the host address for the machine where Discovery is running.
   b. Choose Add Host.
   c. Choose OK.
5. In Step 3 of the wizard, expand the Dynamic Definitions.
   a. Expand HDR.
   b. Right-click Search by Name.... Choose Search for Items.
   c. Type kna1. Choose OK.
   d. Highlight kna1. Choose Next.
   e. Choose Next.
   f. Choose OK.

6. In Notification, choose No.
7. Select ″Open new business object definition to a separate window″. Choose Finish.
8. Save the new business object to the Repository directory.

Preparing the HDR module for testing

To set up the HDR module for testing, use Port Connector:

1. Copy the SAP configuration file. Rename the copied file `portconnector.cfg`.
2. Open `portconnector.cfg` in Connector Configurator.
3. Change the following properties in the Standard tab:
   - `ApplicationName` to `PortConnector`
   - `DELIVERYQUEUE` to `REQUESTQUEUE`
   - `REQUESTQUEUE` to `RESPONSEQUEUE`
4. Save changes.
5. Open `sapconnector.cfg`.
6. Change `REQUESTQUEUE` to `SYNCHRONOUSREQUESTQUEUE`.
7. Save the change.

Testing the HDR module

To test the BAPI module:

1. Open Test Connector.
2. In Business Object Type, select `SAP_kna1`. Choose Create.
3. In Enter Name, type `new object`. Choose OK.
4. Change the verb to Retrieve.
5. Populate `customer_number_KUNNR` with an existing SAP customer number. The number must be 10 digits long, for example: 0000000001.
6. Choose Request > Send.
7. Check the log file for a success message.

Appendix B. Standard configuration properties for connectors

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

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

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

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

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

New and deleted properties

These standard properties have been added in this release.

New properties

- XMLNameSpaceFormat

Deleted properties

- RestartCount

Configuring standard connector properties

Adapter connectors have two types of configuration properties:

- Standard configuration properties
- Connector-specific configuration properties

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

Using Connector Configurator

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

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

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

Setting and updating property values

The default length of a property field is 255 characters.

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

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

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

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

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

Summary of standard properties

Table 51 on page 243 provides a quick reference to the standard connector configuration properties. Not all the connectors make use of all these properties, and property settings may differ from integration broker to integration broker, as standard property dependencies are based on `RepositoryDirectory`.

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

Table 51. Summary of standard configuration properties

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

Table 51. Summary of standard configuration properties (continued)

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

Table 51. Summary of standard configuration properties (continued)

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

Table 51. Summary of standard configuration properties (continued)

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

Standard configuration properties

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

AdminInQueue

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

The default value is CONNECTORNAME/ADMININQUEUE.

AdminOutQueue

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

The default value is CONNECTORNAME/ADMINOUTQUEUE.

AgentConnections

Applicable only if RepositoryDirectory is <REMOTE>.

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

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

AgentTraceLevel

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

ApplicationName

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

BrokerType

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

CharacterEncoding

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

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

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

ConcurrentEventTriggeredFlows

Applicable only if `RepositoryDirectory` is <REMOTE>.

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

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

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

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

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

ContainerManagedEvents

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

The default value is No value.

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

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

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

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

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

ControllerStoreAndForwardMode

Applicable only if RepositoryDirectory is <REMOTE>.

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

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

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

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

The default is true.

ControllerTraceLevel

Applicable only if RepositoryDirectory is <REMOTE>.

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

DeliveryQueue

Applicable only if DeliveryTransport is JMS.

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

The default value is CONNECTORNAME/DELIVERYQUEUE.

DeliveryTransport

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

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

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

WebSphere MQ and IDL

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

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

JMS

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

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

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

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

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

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

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

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

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

DuplicateEventElimination

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

This property can also be set to false.

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

FaultQueue

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

The default value is CONNECTORNAME/FAULTQUEUE.

JvmMaxHeapSize

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

The default value is 128m.

JvmMaxNativeStackSize

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

The default value is 128k.

JvmMinHeapSize

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

The default value is 1m.

jms.FactoryClassName

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

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

jms.MessageBrokerName

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

The default is `crossworlds.queue.manager`.

jms.NumConcurrentRequests

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

The default value is 10.

jms.Password

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

There is no default.

jms.UserName

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

There is no default.

ListenerConcurrency

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

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

Locale

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

A locale name has the following format:

ll_TT.codeset

where:

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

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

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

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

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

LogAtInterchangeEnd

Applicable only if RespositoryDirectory is <REMOTE>.

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

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

MaxEventCapacity

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

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

MessageFileName

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

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

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

MonitorQueue

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

The default value is CONNECTORNAME/MONITORQUEUE

OADAutoRestartAgent

Valid only when the RepositoryDirectory is <REMOTE>.

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

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

The default value is false.

OADMaxNumRetry

Valid only when the RepositoryDirectory is <REMOTE>.

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

The default value is 1000.

OADRetryTimeInterval

Valid only when the RepositoryDirectory is <REMOTE>.

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

The default is 10.

PollEndTime

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

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

PollFrequency

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

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

The default is 10000.

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

PollQuantity

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

PollStartTime

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

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

RequestQueue

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

The default value is CONNECTOR/REQUESTQUEUE.

RepositoryDirectory

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

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

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

ResponseQueue

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

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

RestartRetryCount

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

The default is 3.

RestartRetryInterval

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

The default is 1.

RHF2MessageDomain

WebSphere message brokers and WAS only.

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

A sample header would look like this:

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

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

SourceQueue

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

Designates the JMS source queue for the connector framework in support of guaranteed event delivery for JMS-enabled connectors that use a JMS event store. For further information, see “ContainerManagedEvents” on page 248.

The default value is `CONNECTOR/SOURCEQUEUE`.

SynchronousRequestQueue

Applicable only if `DeliveryTransport` is JMS.

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

The default is `CONNECTORNAME/SYNCHRONOUSREQUESTQUEUE`

SynchronousResponseQueue

Applicable only if `DeliveryTransport` is JMS.

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

The default is CONNECTORNAME/SYNCHRONOUSRESPONSEQUEUE

SynchronousRequestTimeout

Applicable only if DeliveryTransport is JMS.

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

The default value is 0.

WireFormat

Message format on the transport.

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

WsifSynchronousRequest Timeout

WAS integration broker only.

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

The default value is 0.

XMLNameSpaceFormat

WebSphere message brokers and WAS integration broker only.

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

The default value is short.

Appendix C. Connector configurator

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

You use Connector Configurator to:

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

Note:

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

The topics covered in this appendix are:

Overview of Connector Configurator

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

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

You use Connector Configurator to:

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

The mode in which you run Connector Configurator, and the configuration file type you use, may differ according to which integration broker you are running. For example, if WMQI is your broker, you run Connector Configurator directly, and not from within System Manager (see "Running Configurator in stand-alone mode" on page 258).

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 259 to set up a new one.

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

Starting Connector Configurator

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

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

Running Configurator in stand-alone mode

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

To do so:

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

You may choose to run Connector Configurator independently to generate the file, and then connect to System Manager to save it in a System Manager project (see "Completing a configuration file" on page 263.)

Running Configurator from System Manager

You can run Connector Configurator from System Manager.

To run Connector Configurator:

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

To edit an existing configuration file:

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

Creating a connector-specific property template

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

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

- To create a new template, see "Creating a new template" on page 259.
- To use an existing file, simply modify an existing template and save it under the new name.

Creating a new template

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

To create a template:

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

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

     The names of all currently available templates are displayed in the Template Name display.

- To see the connector-specific property definitions in any template, select that template's name in the **Template Name** display. A list of the property definitions contained in that template will appear in the **Template Preview** display. You can use an existing template whose property definitions are similar to those required by your connector as a starting point for your template.

3. Select a template from the **Template Name** display, enter that template name in the **Find Name** field (or highlight your selection in **Template Name**), and click **Next**.

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

Specifying general characteristics

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

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

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

Specifying values

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

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

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

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

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

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

Setting dependencies

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

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

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

   == (equal to)

   != (not equal to)

   > (greater than)

   < (less than)

   >= (greater than or equal to)

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

Creating a new configuration file

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

To select a broker:

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

You can also do this:

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

Creating a configuration file from a connector-specific template

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

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

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

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

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

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

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

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

Using an existing file

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

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

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

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

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

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

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

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

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

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

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

Completing a configuration file

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

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

1. Under the **Standard Properties** tab, select the value field for the `BrokerType` property. In the drop-down menu, select the value `ICS`, `WMQI`, or `WAS`.
2. The Standard Properties tab will display the properties associated with the selected broker. You can save the file now or complete the remaining configuration fields, as described in “Specifying supported business object definitions” on page 266..
3. When you have finished your configuration, click **File>Save>To Project** or **File>Save>To File**.

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

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

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

Setting the configuration file properties

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

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

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

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

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

- Associated Maps
- Resources
- Messaging (where applicable)

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

Standard properties differ from connector-specific properties as follows:

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

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

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

Setting standard connector properties

To change the value of a standard property:

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

Setting application-specific configuration properties

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

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

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

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

Encryption for connector properties

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

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

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

Update method

Refer to the descriptions of update methods found in the *Standard configuration properties for connectors* appendix, under "Setting and updating property values" on page 242.

Specifying supported business object definitions

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

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

If ICS is your broker

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

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

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

To delete a business object from the supported list:

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

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

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

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

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

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

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

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

If a WebSphere Message Broker is your broker

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

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

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

If WAS is your broker

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

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

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

Associated maps (ICS only)

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

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

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

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

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

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

- **Business Object Name**

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

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

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

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

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

  To explicitly bind a map:

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

Resources (ICS)

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

Messaging (ICS)

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

Setting trace/log file values

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

To change the logging and tracing values:

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

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

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

Data handlers

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

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

Saving your configuration file

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

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

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

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

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

Changing a configuration file

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

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

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

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

Completing the configuration

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

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

Using Connector Configurator in a globalized environment

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

Connector Configurator supports non-English characters in:

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

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

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

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

Appendix D. Troubleshooting the connector

The appendix describes problems that you may encounter when starting up or running the connector component of the Adapter Guide for mySAP.com (R/3 V.3.x).

This chapter contains the following sections:

Generic troubleshooting

This section describes problems that you may encounter when starting up or running any module of the IBM WebSphere Business Integration Adapter for mySAP.com. It covers three troubleshooting areas:

Startup problems

The following subsections provide suggestions for common startup problems.

Connector fails to start

If you encounter difficulties when trying to start the connector:

- Check to make sure that the integration broker is up and running.
- Check that the SAP application is running.
- Verify that the standard and connector-specific configuration properties are set properly. For more information, see "Configuring the connector" on page 17, and Appendix D, "Troubleshooting the connector," on page 273.

Connector cannot log on to the SAP application

If the connector cannot log on to the SAP application:

- Check that the SAP application is available.
- Ensure that you have properly set the standard and connector-specific connector configuration properties by checking the `Sysnr`, `Client`, `Hostname`, and `Modules` properties. For more information, see "Configuring the connector" on page 17 and Appendix B, "Standard configuration properties for connectors," on page 241.
- Verify that the user name and password set up for the connector has the appropriate level of privileges.

Connector logs on and the session closes

If the connector successfully logs on to the SAP application and then the session closes immediately, there may be a database problem. Check that `PSAPUSER1D` and `PSAPUSER1I` tablespaces have sufficient space allocated to them. By default, the SAP system provides minimal space for these two tablespaces. The connector requires more than the default amount of space. For more information, see "Increasing log tablespace size" on page 54..

Note: This problem is relevant to all connector modules except the RFC Server Module.

Connector dies

If the connector dies with a message "`connection to the SAP application is lost`"or you get an RFC system exception, then you may have a network problem. Check the short dump for the connector user or the time when the error occurred. Use the IBM CrossWorlds Station tool or go to transaction ST22. If you still need more information, check the system log by going to transaction SM21.

Default values are not being set

Default values have been set in a business object but the connector is not picking up the values. This is a configuration issue. For default values to be used, the `UseDefaults` connector property needs to be set to true and each attribute requiring a default value needs to be marked as required in the business object definition.

Collaborations not subscribing to business objects (WebSphere InterChange Server only)

If a collaboration is not subscribing to a particular business object on a specified WebSphere InterChange Server, then:

- Check that the collaboration is configured to subscribe to that particular business object.
- Verify that the collaboration is running.
- Verify that the map references have the correct business object specified as the source business object.

Encoding of binary data (message brokers only)

For fields with binary data (RAW data type in an SAP system), the adapter will encode the value for the fields in hexadecimal rather than the more typical base64 encoding in the XML MQ message. As well, the adapter also expects data from a service call request to be in hexadecimal encoding in the XML MQ message.

WBI performance tuning and memory management

Java Virtual Machines (JVMs) externalize multiple tuning knobs which may be used to improve WebSphere Business Integration application performance. These knobs control issues related to garbage collection, heap size, threading, and locking. Because the ICS server and its components (maps, collaborations) as well as most of the adapters are written in Java, the performance of the JVM has a significant impact on the performance delivered by an ICS application.

This section addresses potential issues with garbage collection, heap size, and thread stack size. The following URL provides a useful summary of JVM options: http://java.sun.com/docs/hotspot/VMOptions.html

The following URL provides a useful FAQ about the HotSpot Engine: http://java.sun.com/docs/hotspot/PerformanceFAQ.html#20

For a detailed description of the IBM JVM the reader should consult the Java Performance issue of the "IBM Systems Journal", Vol. 1, 2000: http://www.research.ibm.com/journal/sj39-1.html

Setting heap size quick start

- Make sure that the heap never pages.
- To optimize memory usage, analyze `verbose:gc` trace.
- Aim for less than 10% execution time in GC.
- For optimal performance, the heap should be run with 50% - 60% occupancy.
- Avoid finalizers.
- Avoid compaction.
- Analyze requests for large memory allocations and devise a method for reusing the object.

Setting the heap and nursery size for garbage collection

Garbage collection is the process of freeing unused objects in the JVM so that portions of the heap can be reused.

Garbage collection occurs when there is a request for memory, and the request cannot be readily satisfied from the free memory available in the heap (allocation failure). Garbage collection also occurs when a Java class library `System.gc()` call is made. In this case garbage collection occurs immediately and synchronously.

While the function provided by the SUN HotSpot and IBM garbage collectors is the same, the underlying technology is different. For both JVMs garbage collection takes place in three phases: mark, sweep, and an optional compact phase.

The implementation of the garbage collection phases is different because the Sun HotSpot engine is a generational collector and the IBM JVM is not. A detailed discussion of the HotSpot generational collector can be found at the following URL: http://java.sun.com/docs/hotspot/gc/index.html

With the IBM JVM, the full heap is consumed before a garbage collection is triggered. The first phase is to mark all referenced objects in the region being collected, which leaves all un-referenced objects unmarked and the space they occupy free to be collected and reused. Following the mark phase, free chunks of memory are added to a freelist. This phase is referred to as sweeping. For performance reasons, the IBM JVM only frees chunks of heap space greater than 512 bytes.

Following the sweep phase, a compact phase is sometimes performed. The compact phase moves objects closer together to create larger contiguous free chunks. Because compaction is time-consuming, avoid it when possible For most `System.gc()` calls compaction is performed. The IBM JVM has been optimized to avoid compaction.

The following table explains which phases of garbage collection are multi-threaded and which are concurrent. Concurrent means the process runs while the application threads continue to execute. If the process is not concurrent it means that the program pauses during that phase of garbage collection.

Table 52. JVM Release Mark Sweep Compact

JVM release	Type	Mark	Sweep	Compact
Sun HotSpot 1.3.1	Multithreaded	No	No	No
Sun HotSpot 1.3.1	Concurrent	No	No	No
IBM JVM 1.3.1	Multithreaded	Yes	Yes	No
IBM JVM 1.3.1	Concurrent	Optional	No	No

Monitoring garbage collection

A `verbosegc` trace prints garbage collection actions and statistics to `stderr`. The `verbosegc` trace is activated by using the Java run time option of `-verbose:gc`. Output from `-verbose:gc` is different for the Sun HotSpot and IBM JVMs. Below are sample output from a `verbosegc` trace with embedded explanations of key pieces of information for both an IBM JVM and Sun HotSpot.

IBM JVM -verbose:gc output

```
<AF[8]: Allocation Failure. need 1572744 bytes <-amount of memory
requested, 5875 ms since last AF> <AF[8]: managing allocation failure,
action=1 (23393256 <-free at alloc failure)/131070968 <- heapsize)
(2096880/3145728)> <GC: Tue Dec 18 17:32:26 2001 <GC(12): freed 75350432
bytes in 168 ms <- duration of GC, 75% free (100840568 <-free)/134216696 <-
total heapsize)> <GC(12): mark: 129 ms, sweep: 39 ms, compact: 0 ms
<-compact did not run> <GC(12): refs: soft 0 (age >= 32), weak 0, final 0
<-no finalizers, phantom 0> <AF[8]: completed in 203 ms>
```

SUN JVM -verbosgc (young and old)

```
[GC 325816K->83372K(776768K), 0.2454258 secs <-duration of GC] [Full GC
267628K->83769K <- live data (776768K <-size of heap), 1.8479984 secs]
```

Setting the heap size for most configurations

This section contains guidelines for determining the appropriate Java heap size for most WBI configurations.

For many applications, the default heap size setting for the IBM JVM is sufficient for good performance. In general, the HotSpot JVM default Heap and Nursery sizes are too small. To set the optimal heap size for the IBM JVM on AIX, follow these guidelines.

In order to effectively use rate-trigger heap growth just set the `-ms` to 64MB or 96MB, and the `-mx` to 256-512MB. Ensure that `-mx` does not force the heap to page. The JVM will try to control the GC time by growing and shrinking the heap. The output from `-verbose:gc` monitors the GC actions.

A similar process can be used to set HotSpot heaps. In addition to setting the minimum and maximum heap size, one should also increase the Nursery size to approximately 1/4 of the heap size.

Note: Note that one should never increase the Nursery to more than 1/2 the full heap.

The nursery size is set using the `MaxNewSize` and`NewSize` parameters (i.e., `-XX:MaxNewSize=128m, -XX:NewSize=128m`).

Once the heap sizes are set, `verbose:gc` traces monitor the GC actions. If the percentage of time in GC is too high and the heap has grown to its maximum, increase `-mx`.

Note: This setting will not always solve the problem, which is normally a memory over-usage problem. If pause time is too long then decrease heap size. If both problems are observed, analyze the application heap usage.

Setting heap size when running many JVMs on one system

Each running Java program has an associated heap. If the sum of all of the Java heap sizes and all other usages of virtual memory exceeds the size of physical memory, then the heap page, causing performance to degrade. To minimize the paging, use the following guidelines:

- For a trial run, activate `verbosegc` for each running JVM.
- Based on the `verbosegc` output, set the initial heap size to a relatively low value. For example, assume that the `verbosegc` report shows that the heap size grows quickly to 32 MB, then grows more slowly to 40 MB. Based on this, set the initial heap size to 32 MB (-Xms32m).
- Based on the `verbosegc` output, set the maximum heap size large enough to allow for peak throughput. In the previous example, a maximum heap size of 64 MB may be appropriate (`-Xmx64m`).
- Do not set the heap sizes too low, so that garbage collections do not occur too often but large enough to avoid paging.

Reducing or increasing heap size if java.lang.OutOfMemoryError occurs

The `java.lang.OutofMemoryError` is used by the JVM in a variety of circumstances. The exception occurs if there is not enough heap space for an object in the heap, or if other resources outside the Java heap have been exhausted.

Read the output from `java.lang.OutofMemoryError` to see if the problem is due to a lack of memory in the heap. If so, increase the size of the heap.

If the heap appears to be large enough, check the finalized count from the `-verbose:gc`. If the count appears high, resources outside the heap might be held by objects within the heap and cleaned by finalizers. Reduce the size of the heap and increase the frequency with which finalizers are run.

Setting AIX threading parameters

The IBM JVM threading and synchronization components are based upon the AIX Posix compliant Pthread implementation. The following environments variables have been found to improve Java performance in many situations and have been used for the benchmarks in this document. The variables control the mapping of Java threads to AIX Native threads, turn off mapping information, and allow for spinning on Mutex locks.

- export `AIXTHREAD_COND_DEBUG=OFF`
- export `AIXTHREAD_MUTEX_DEBUG=OFF`
- export `AIXTHREAD_RWLOCK_DEBUG=OFF`
- export `AIXTHREAD_SCOPE=S`
- export `SPINLOOPTIME=2000`

More information on AIX specific Java tuning information can be found at: http://tesch.aix.dfw.ibm.com/java/perftips.html

Using HotSpot Server instead of Client

The Sun HotSpot JVM can be configured to run as a server or as a client. When configured as a server the JIT (Just-In-Time Compiler) uses extra processor cycles and memory to create more highly optimized code. Since the ICS is a long running process the extra time and memory spent JITting at initial instantiation is well worth the increased performance during run time.

Therefore, when using the Sun HotSpot JVM, the ICS should always be run as a server. To do this, the –server parameter is added to the invocation of the ICS process.

Setting thread stack size if using many threads

As mentioned in the section on ICS threading, Java threads consume memory in the heap. In addition, the threads themselves use virtual memory for their thread stacks. If a configuration is using an excessive number of threads, memory in either place may become a problem. The JVM allows a user to configure the amount of virtual memory set aside for the thread stack. The default thread stack size is different depending on the JVM version and the operating system. However, the mechanism to set the value is the same. To set the thread stack size to 128KB, the parameter -ss128k is passed in on the invocation of the JVM. Care should be taken not to set this value to small. It is recommend that at least 128KB be given to each thread stack, although the system may operate successfully with a lower setting.

SAP notes about memory management

Refer to the following SAP notes for resolving memory related issues:

- SAP Note 558250: Memory problems with SAP Java Connector
- SAP Note 634689: Central Note for Memory Issues

Troubleshooting for the ABAP Extension Module

This section describes problems that you may encounter when starting up or running the ABAP Extension Module. It covers three troubleshooting areas:

- "Transport files"
- "Startup problems" on page 279
- "Event handling" on page 280
- "Event distribution problem on Microsoft Windows (connector version 4.2.7 only)" on page 279

Transport files

If you get errors when installing the adapter's transport files for the ABAP Extension Module:

- Verify that you installed the correct transport files. IBM WebSphere Business Integration Adapter for mySAP.com You must install version 3.x transport files on an R/3 version 3.x system and version 4.x transport files on an R/3 version 4.x system. Transport files are installed in their own directories (`transports_3x` and `transports_4x`).
- Verify that you installed the transports in the correct order. Some transport files have dependencies such as existing tables.

For example, one transport file creates a data element for a table and another transport creates a table for that data element. If the table is not created first, the system returns an error.

- Verify that all of the necessary transport files were installed properly. Each transport file adds specific functionality for the connector. For example, IBM CrossWorlds Station is separated into two transports (`_Tools_Development` and `_Tools_Maintenance`). The two transport files enable you to tailor your installation so that you do not have to install all of the IBM CrossWorlds Station tools on your systems. For example, you may not want to install development tools in a production environment.

  You must install at least the Primary, Utilities, Request, and Delivery transport files. For more information, see "Connector transport files" on page 48..

Startup problems

If the connector logs in to the SAP application successfully, but the connector's log in the SAP application is empty:

- Check that logging is turned on. If logging is turned off, use IBM CrossWorlds Station to turn it on. By default, logging is set to 1. For more information, see Chapter 9, "Managing the ABAP Extension module," on page 107.
- Check that the connector is logged on to the same machine where you are viewing the connector log file.
- Check that the `Namespace` configuration property is set to `true`. If you have upgraded to the connector's namespace from the previous `YXR` environment, the connector may still be logging into the `YXR` environment. If this is the case, set the `Namespace` configuration property to `true`. For more information, see the "Namespace" on page 23 property in the "Connector-specific configuration properties" on page 18.
- Check that the number range in the connector log is in sync. If you have upgraded the NumberRange transport number, then number range intervals may be out of sync. Verify that the number range object number is lower than the first number in the connector log.

  To check the number ranges, go to transaction SNRO and enter `/CWLD/LOG` in the Number Range Object field. Click the Number Ranges button, click the Display Intervals button, and note the number range object number. Open the connector log and note the number of the first entry. If this number is higher than the number range object number, then the log entry number in the connector log needs to be modified to be one number higher. For more information, see "Verifying number ranges for transport objects" on page 54..

Event distribution problem on Microsoft Windows (connector version 4.2.7 only)

After upgrading to the IBM CrossWorlds Connector for SAP Version 4.2.7 on Windows, events remain in the event table and are not picked up and processed by the connector in the following circumstances:

- You configured event distribution across multiple connectors.
- The connector is running against an SAP 3.x system that loaded the NON-namespace (`yxr`).

This problem is caused by a change SAP has made in their java API (SAPJCo).

To fix the problem, load a patch transport that changes only the event request and event return function modules provided by the IBM WebSphere Business

Integration Adapter for mySAP.com. Load this patch transport in 4.0 and 4.5 SAP systems that do *not* have the namespace (/CWLD/) infrastructure.

Note: The namespace ABAP infrastructure does not have this problem.

Event handling

The following subsections provide suggestions for event handling problems.

ABAP Extension Module is not invoked by subscribing business objects

If a subscribing business object is not being processed by the ABAP Extension Module, then:

- Check that the vision connector framework is set to call the ABAP Extension Module. The Modules property must be set to Extension.
- Check that the connector subscribes to the business object.

Connector is not picking up events

If your connector is not picking up events from the SAP application:

- Check the connector's event table in the SAP application to see if the event is queued for your connector.
- In a multiple connector environment, if the event is not queued, make sure there is an entry in the Event Distribution table (/CWLD/EVT_DIS) for the combination of your connector and business object. Check to see that this combination is unique.

  If you have multiple connectors subscribed to the same business object, then one connector might be processing the wrong events. For more information on distributing events between multiple connectors, see on page 44..
- If you have a lock object for an event in the SAP application, then the SAP application may not finish processing the save process for that event.

  Check the connector's event table in the SAP application to see if the event has a status of L (Locked). If the status is L, then you most likely have a problem in the SAP application and not the connector.
- The connector might have died while processing the event. Check the status of the event in the connector's event table in the SAP application. If the status is R (Retrieved), then the event has not been moved to the archive table. If the event's status is R, verify that the event did not make it to the destination.

  If the event did not make it to the destination, change the status from R to Q (Queued). Events with a status of Q are picked up by the connector at the next poll interval. To change the status from R to Q, go to the event table, select the event, and then click the Edit button. In the window that appears, change the Event Status field from R to Q.

Business object fails to process

If a business object fails to process successfully, check the connector log in the SAP application. Entries for failed events appear in red. Reprocess events using the reprocessing tool, which enables you to set breakpoints in the code as you step through the transaction.

Attention: Do not use the reprocessing tool in a production environment, because it causes the WebSphere business integration system and the SAP application to be out of sync.

Deadlock with Event Table

The current event table and the future event table, may encounter a deadlock situation if there are many events being added at one time. This situation occurs if the provided indices for the event table are not used because of database tuning. Tuning normally occurs during off-peak hours when there are few or no events in the event table. When a database table has no or few entries, it is more efficient not to use an index for reading the table. To avoid a deadlock situation, exclude the current event and future event tables when running a database tuning utility.

Large Objects

Large objects may require additional changes to process successfully. ABAP Extension Module objects are converted to a flat structure before passing the data to the SAP application or converted from a flat structure when receiving the data from the SAP application. See "Business object conversion to a flat structure" on page 58 for more information. This flat structure is held in memory with each attribute for an object instance being a row in the structure. For each attribute, 373 bytes of data are passed between the connector and the SAP application. The number of attributes multiplied by 373 gives an approximation of the size of the flat structure. As well, an instance of the object is also in memory. Therefore, an object with many child objects (segments) may require a change to the Java heap size in the startup script for the connector's Java process in order to avoid an out-of-memory error.

Windows

In the `start_SAP.bat` script, change the `-mx128m` Java heap size options parameter default value to a value large enough to handle the flat structure and the instance of the object. A value larger than the available memory on the machine running the Java process will also result in an out-of-memory error. The `128m` represents a maximum Java heap size of 128 MB.

Unix:

The SAP application may also require changes to the ABAP timeout parameter to process a large object successfully.

Troubleshooting for the BAPI module

This section describes problems that you may encounter when running the BAPI module.

Event handling

The following subsections provide suggestions for common event handling problems.

BAPI module is not invoked by subscribing business objects

If a subscribing business object is not being processed by the BAPI module, then:

- Check that the vision connector framework is set to call the BAPI module. The `Modules` property must be set as follows: `Bapi`.
- Check that the connector subscribes to the business object.
- Check that the SAPODA-generated BAPI-specific business object handler class file is in the `\bapi\client` directory. If the class file is not in this directory, then

the BAPI business object handler is not invoked to process the business object. For more information, see “Using generated business objects and business object handlers” on page 199..

- Check that the BAPI business object handler name in the business object verb application-specific information is correct. For more information, see “Business object application-specific information” on page 174..
- Ensure that when you generated the business object handler, you specified the appropriate verb to match the BAPI.

Business object fails to process

If a business object fails to process successfully:

- Check that the BAPI you are using has a return business object. The BAPI module looks in the return business object for messages with the key `e` (error) or `a` (abort). If the module finds one of these keys, then it notes that the event has failed. If the BAPI does not have a return business object, make sure you implement your own error handling.
- Use transaction SE37 to test the BAPI associated with the failed event. This should enable you to reproduce the failure.

  If this does not work, then you may have a problem in the conversion from internal formats to external formats. Check that you are specifying values in the correct format. For example, for dates, SAP's internal format is YYYYMMDD and you may be specifying the format MMDDYYYY. This causes the BAPI to fail, because the specified format is not understood.
- Check that the application-specific information of each attribute is correct. If these values are not correct, then the BAPI module does not populate the object correctly before sending it back to the SAP application.
- Check that the I and E parameters are specified properly. Remember that `I` identifies the import parameter and `E` identifies the export parameter. For more information, see “Business object fails to process” on page 283.

Connector appears to be polling but events are not being picked up

The BAPI module includes a dummy implementation of the `pollForEvents()` method. The connector appears to be polling because it returns a polling message. The BAPI module does not support polling, so ignore these messages.

If you want to implement polling for the BAPI module, you must use the polling capabilities in the ABAP Extension Module. For more information, see Chapter 3, “Overview of the ABAP Extension module,” on page 33..

Troubleshooting for the RFC Server Module

This section describes problems that you may encounter when starting up or running the RFC Server Module. It covers:

Startup problems

If the connector cannot register with the SAP application:

- Check that the SAP application is available.
- Check that you have properly set the standard and connector-specific connector configuration properties. Specifically, check the `gwService,, Hostname,,`

RfcProgramId,, and Modules, properties. For more information, see "Configuring the connector" on page 17 and Appendix D, "Troubleshooting the connector," on page 273.

Connector dies

If your connector dies, check the following:

- Check that threads are being spawned by the RFC Server Module. Verify that the NumberOfListeners property is set properly. For more information, see the "NumberOfListeners" on page 23..
- Verify that the RFC program ID is set up so that the RFC Server Module registers itself with the SAP Gateway. For more information, see the "RfcProgramId" on page 23 and "Registering the RFC Server Module with the SAP gateway" on page 189..

Event handling

The following subsections provide suggestions for common event handling problems.

RFC Server Module is not invoked by subscribing business objects

If a subscribing business object is not being processed by the RFC Server Module, then:

- Check that the vision connector framework is set to call the RFC Server Module. The "Modules" on page 23 property must be set as follows: RfcServer.
- Check that the connector subscribes to the business object.
- Check that the SAPODA-generated BAPI-specific business object handler class file is in the \bapi\server directory. If the class file is not in this directory, then the BAPI business object handler is not invoked to process the business object. For more information, see "Using generated business objects and business object handlers" on page 199.
- Check that the BAPI business object handler name in the business object verb application-specific information is correct. For more information, see "Business object fails to process" on page 280.
- Check that the specified verb for your BAPI-specific business object handler is correct for the type of processing you need. Specifically, make sure that when you generated the business object handler, you specified the appropriate verb to match the BAPI.

Business object fails to process

If a business object fails to process successfully:

- Check that the BAPI you are using has a return business object. The RFC Server Module looks in the return business object. for messages with the key e (error) or a (abort). If the module finds one of these keys, then it notes that the event has failed. If the BAPI does not have a return business object., make sure you implement your own error handling.
- Use transaction SE37 to test the BAPI associated with the failed event. This should enable you to reproduce the failure.

  If this does not work, then you may have a problem in the conversion from internal formats to external formats. Check that you are specifying values in the correct format. For example, for dates, SAP's internal format is YYYYMMDD and you may be specifying the format MMDDYYYY. This causes the BAPI to fail, because the specified format is not understood.

- Check that the application-specific information of each attribute is correct. If these values are not correct, then the RFC Server Module does not populate the object correctly before sending it back to the SAP application.
- Check that the I and E parameters are specified properly. The I parameter identifies the import parameter and the E parameter identifies the export parameter. For more information, see "Business object fails to process" on page 282.

Troubleshooting for the ALE Module

This section describes problems that you may encounter when starting up or running the ALE Module. It covers the following subjects:

Startup problems

The following subsections provide suggestions for common startup problems.

Connector cannot log on to or register with the SAP application

If the connector cannot log on to or register with the SAP application:

- Check that the SAP application is available.
- Check that you have properly set the standard and connector-specific connector configuration properties:
  - Check that the required MQSeries queues have been created and that their corresponding configuration property correctly specifies their name.
  - For request processing, check the Sysnr, Client, Hostname, and Modules properties.
  - For event processing, check the gwService, Hostname, RfcProgramId, and Modules properties.

  For more information, see "Configuring the connector" on page 17 and Appendix D, "Troubleshooting the connector," on page 273.
- Verify that the user name and password set up for the connector has the appropriate level of privileges.

Connector is not polling events

If your connector is not polling events from the SAP application:

- Check that the verb application-specific information for the desired verb has been modified to have the correct message type, message code, and message function.
- Check that the verb AleOutboundVerbs exists and has a list of valid verbs.

Connector appears to be polling but events are not being picked up

- Check that the event queues (`SAPALE_Event_Queue` and `SAPALE_Wip_Queue`) have been created correctly and that polling is being done on the `event` queue.
- Verify that the following are running on your system:
  - MQSeries
  - TCP/IP

- Verify that the ALE configuration within the SAP application is correct; for more information see, Chapter 11, "Overview of the ALE module," on page 125.
- Check that the connector has made at lease one poll call; doing so installs the function modules for event processing.
- Check that a message has been written to the `wip` queue and has been moved to the `event` queue.

Event handling

The connector logs information about successfully processed IDocs in a JMS-MQ event message (in the queue specified in the `SAPALE_Event_Queue` configuration property) to the `EventState.log` file. This file is located in the directory specified in the `AleEventDir` configuration property.

Note: Each event message can contain multiple IDocs, each of which represents a business object.

If the connector goes down before it processes all IDocs in the current event message, it uses the `EventState.log` file during recovery to ensure that it sends each IDoc only once to the integration broker.

Important: The connector does not create the log file automatically the first time it processes an event. You must create this file for before you run the connector for the first time.

The format of the log file is:

TID: 0S, 1S, 2F, 3U

where <*TID*> is the current transaction ID being processed, and each number represents the sequence number of all work units in the event message.

For example, if the connector has successfully processed three of the first four IDocs in the current event message, the second IDoc failed processing, and the connector has not yet finished processing the current event message the `EventState.log` file might show:

<TID> :: 0S, 1F, 2S, 3S

If the connector went down before processing the entire event message, at startup the connector will use the information in the log file to resume processing the events in the message at the point where it had stopped processing. The connector reads the log to get the transaction ID of the event to be recovered, the latest work unit, and the status of each work unit. Then the connector begins sending to the integration broker the business objects that represent each IDoc in the event message with a sequence number greater than the last number in the log file. In the previous example, the connector will processing the fifth IDoc in the current event message.

The connector keeps the contents of the log file in memory to enhance performance. It accesses the file on disk only to update it with a new entry. The connector reads the log file only at recovery time.

For information on how the connector uses the `EventState.log` file in the recovery process, see "Failure recovery" on page 286.

Failure recovery

Note: These recovery steps do not apply if a disk failure occurs or if a disk is full.

To recover from failures during event notification, the connector:

1. The connector processes IDocs from the JMS-MQ message in the `event` queue (specified in the `SAPALE_Event_Queue` configuration property). When it successfully processes a IDoc, the connector logs an entry in the `EventState.log` file.
   - If none of the work units in the event message fails processing, the connector moves the event message to the archive queue with an `IDocProcessStatus` value of success.
   - If any of the work units in the event queue message fails processing, the connector will move the event message to the archive queue and update the `IDocProcessStatus` value of partial.
2. After the connector processes all IDocs in an event message, it clears the `EventState.log` file and begins writing entries to it for the next event message.
3. If the connector goes down before it processes all IDocs in an event message, it uses the information in `EventState.log` to determine where to begin processing during the recovery process. When it comes back up, the connector checks whether there are any entries in the log file.
   - If there are no entries, the connector sends all IDocs in the event message to the integration broker.
   - If there are entries, the connector will use this information to resume processing an event message at the point where it had stopped processing. The connector reads the log to get the name of the event message to be recovered and the latest IDoc sequence number. Then the connector sends to the integration broker each IDoc in the event message with a sequence number greater than the last number in the log file. In this example, the event message is moved to the archive queue and the `IDocProcessStatus` is updated according to the status of each work unit in the `EventState.log`.

     Using the log file prevents the connector from sending the same IDoc multiple times to the integration broker. The connector keeps the log file in memory to enhance performance. The connector accesses the file on disk only to update it with a new entry, and reads the log file only at recovery time.

     Note: If there is no IDoc in the event message whose sequence number is greater than the last number in the log file, the connector went down after processing the last event but before archiving the event file. In this case, the event message is moved to the archive queue and the `IDocProcessStatus` is updated according to the status of each work unit in the `EventState.log`.

Recovery from business object creation errors

If the connector has created only the header portion of the message in the WIP queue but not the data portion, this procedure will recover the data portion of the message.

1. Examine the SAP connector log for error messages pertaining to the business object's name, message type, or verb.
2. Make the appropriate corrections to the business object definition or the connector configuration.

Note: Configuration changes may include changes to MQSeries queues. For more information, see "Prerequisites to running the ALE Module" on page 131..

3. Restart the connector.

Request processing

If a subscribing business object is not being processed by the ALE Module, then:

- Check that the vision connector framework is set to call the ALE Module. The Modules property must be set to `ALE`.
- Check that the connector subscribes to the business object.

Troubleshooting the Hierarchical Dynamic Retrieve Module

This section describes problems that you may encounter when starting up or running the Hierarchical Dynamic Retrieve Module. It covers the following areas:

- "Error handling and logging"
- "SQL SELECT fails" on page 288

Error handling and logging

The connector logs an error message when it encounters a condition that causes the retrieval to fail. When such an error occurs, the connector also prints a textual representation of the failed business object as it was received from the integration broker. It writes the text to the connector log file or the standard output stream, depending on its configuration. You can use the text to find the source of the error.

Error types

Table 53 describes the types of tracing messages that the Hierarchical Dynamic Retrieve Module 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 WebSphere MQSeries message interface.

Table 53. Connector tracing messages

Tracing level	Tracing messages
Level 0	Message that identifies the connector version. No other tracing is done at this level.
Level 1	Function module entry and exit messages. These messages are written whenever the connector execution thread enters or exits from a function. The messages help to trace the process flow of the connector.
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
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, finds a business object in an array of child business objects, or removes child business objects during a retrieval.

Table 53. Connector tracing messages (continued)

Tracing level	Tracing messages
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 properties • Messages that identify when the connector enters or exits a Java method, which helps trace the process flow of the connector • SQL statements. At this level and above, the connector prints out all SQL statements that it executes. • Changes to an attribute value during a retrieve. At this level and above, the connector prints out the name of the attribute and its new value.
Level 5	• Messages that indicate connector initialization, for example, messages showing the value of each configuration property retrieved from the integration broker • Messages that comprise a business object dump • Messages that comprise a representation of a business object before the connector begins processing it (displaying its state as the connector receives it from the integration broker) and after the connector has completed its processing (displaying its state as the connector returns it to the integration broker)

Connector message file

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

```
1210

SAP Hierarchical Dynamic Retrieve module unable to initialize.

1211

SAP Hierarchical Dynamic Retrieve module failed to locate...
```

Fails to call RFC_READ_TABLE

The SAP `RFC_READ_TABLE` function doesn't handle character-based datatypes. The module may fail while retrieving data if the fields use the following datatypes:

- CURR
- DEC
- FLTP
- INT1
- INT2
- INT4
- LRAW
- RAW
- RAWSTRING

SQL SELECT fails

If a `SELECT` statement fails, check whether any simple attribute that is marked as key or is used as a foreign key contains a single quotation mark (’). If so, revise the business object's map to convert the single quotation mark (’) to two single quotation marks (’’).

Troubleshooting SAPODA

There are two known problems you might encounter when using SAPODA:

- SAPODA runs without messages.

  If the message file specified for the ODA does not exist, the ODA runs without messages. This problem is caused during configuration of the ODA when Business Object Designer displays a default name for the message file. The default name follows the naming convention:

  *AgentName*Agent.txt

  If the name of the actual message file does not follow this convention and the default value is not overwritten with the actual value, Business Object Designer displays an error message in the window from which the ODA was launched. This message does not pop up in Business Object Designer.

  For more information, see“Working with error and trace message files” on page 295..

- On a Windows system, if Business Object Designer cannot find required library files in the Path environment variable or the files are not on the system, it displays a CORBA Exception while attempting to get the tree nodes. For information about these files, see“Before using SAPODA” on page 291..

Appendix E. Generating business object definitions using SAPODA

This chapter describes SAPODA, an object discovery agent (ODA), which generates business object definitions for the IBM WebSphere Business Integration Adapter for mySAP.com (SAP R/3 Version 4.x). Because the connector works with objects that are based on IDoc types, BAPIs, and RFC-enabled function modules defined in an SAP system, SAPODA uses these objects to discover business object requirements specific to its SAP data source.

Note: Familiarity with IDoc types, BAPIs, and RFC-enabled function modules within an SAP system can aid in understanding how SAPODA operates.

Installation and usage

This section discusses the following:

Installing SAPODA

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

- `ODA\SAP\SAPODA.jar`
- `ODA\messages\SAPODAAgent.txt`
- ODA\messages\SAPODAAgent_*ll*_*TT*.txt (message files specific to a to a language (_*ll*)country or territory (_*TT*))
- `ODA\SAP\start_SAPODA.bat` (Windows only)
- `ODA/SAP/start_SAPODA.sh` (UNIX only)

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

Before using SAPODA

This section describes the following:

Before running SAPODA

Before you can run SAPODA, you must:

- Have a valid logon ID to the SAP system
- Have downloaded the SAP Java API, which SAP calls their Java Connector (SAPJCo).
  - If you have already followed instructions for installing the IBM WebSphere Business Integration Adapter for mySAP.com on the same machine on which you install SAPODA, you should have already completed this step.
  - If you are installing SAPODA prior to installing the adapter or on a different machine, you must download the SAP Java Connector.

    Make sure that you download the SAPJCo for the operating system that your connector is running on. The SAPJCo is available for download from SAP's website at `http://service.sap.com/connectors`. You must have a SAPNet account to access the SAPJCo (if you do not already have one, contact your local SAP BASIS administrator).
- Have copied the following SAPJCo files to the appropriate directory and verified that they are in the Path environment variable.

  Important: If you cannot find these SAPJCo files for an earlier version of SAPJCo than 1.1, contact technical support.

Unix

- `jCO.jar`
- `librfccm.so (AIX) or librfccm.so (Solaris)`
- `jCO.jar librfccm.so(LINUX)`
- `libjRFC11.so`
- `libjRFC12.so`

If you have already followed instructions for installing the IBM WebSphere Business Integration Adapter for mySAP.com on the same machine on which you install SAPODA, copy these files from the `/connectors/SAP` directory to the `/ODA/SAP` directory. If you install SAPODA on a different machine from the connector, after you unzip the SAPJCo file, copy these four files to the `/ODA/SAP` directory.

Windows

- jCO.jar
- librfc32.dll
- jRFC11.dll
- jRFC12.dll

If you have already followed instructions for installing the IBM WebSphere Business Integration Adapter for mySAP.com on the same machine on which you install SAPODA, copy these files from the \connectors\SAP directory to the \ODA\SAP directory. If you install SAPODA on a different machine from the connector, after you unzip the SAPJCo files, copy these four files to the \ODA\SAP directory.

For Windows, the librfc32.dll requires one or more C runtime dlls. The C runtime dlls depend on the version of the SAP release being used. Through SAP release 45B, the C runtime dll required is msvcrt.dll version 5.00.7022 or newer. Starting with SAP release 46A, the C runtime dlls required are msvcrt.dll version 6.00.8267.0 or newer and msvcp60.dll version 6.00.8168.0 or newer. The dll or dlls should be copied into the C:\WINNT\system32 directory. This dll or these dlls may already be present and if not, can be found on the "Presentation CD" that contains the Windows SAPGUI setup in the folder <*cddrive*>:\GUI\Windows\Win32\system. See SAP OSS note number 0182805 for more information.

Before using SAPODA

Before using SAPODA to generate a business object definition from an SAP Intermediate Document (IDoc) format, you must create the IDoc definition file for each IDoc you want supported. SAPODA uses this file as input.

How to use SAPODA

After installing SAPODA, 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 SAPODA

You can launch SAPODA in either of the following ways:

- If you registered SAPODA with the Object Activation Daemon, you do not need to manually start SAPODA. OAD maintains a list of registered ODA names, and listens for requests to start the ODA. When you select the ODA's name in Business Object Designer, OAD starts the ODA.

  For information on registering SAPODA, see the installation guide for your WebSphere business integration system.

- If you have not registered SAPODA with the Object Activation Daemon, start it by running the appropriate file:

UNIX

```
start_SAPODA.sh
```

End of UNIX

Windows

```
start_SAPODA.bat
```

End of Windows

Important: If you register the ODA with OAD, but run the script or batch file before selecting the ODA in Business Object Designer, Business Object Designer may display two names when you press Find Agents: the one registered with OAD and the one in the script or batch file. If the two names are identical, two identical names display. However, if both names represent the ODA in the same subnet and the ODA has been started manually, Business Object Designer connects only to the manually started ODA. In this case, Business Object Designer does not call OAD. Attempting to start the second identically named ODA in the same subnet causes the second ODA to quit with an error. For information on changing the name of an ODA, see "Running SAPODA on multiple machines" on page 294.

You configure and run SAPODA 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 `SAPODA`. 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 SAPODA on multiple machines

It is recommended that you change the name of the ODA when you run multiple instances of it on different machines. To create additional uniquely named instances of SAPODA, specify a unique name in the `AGENTNAME` variable of the script or batch file on each machine on which the ODA is installed.

To run the desired ODA, you select it by name from a list of available ODAs in Business Object Designer, which displays all active ODAs. Non-unique names can cause confusion when selecting the appropriate ODA to run.

A naming convention for assigning unique names is prefixing each name with the name of the host machine on which the ODA runs. 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 34 on page 297 illustrates the window in Business Object Designer from which you select the ODA to run.

Working with error and trace message files

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

*AgentName*Agent_ll_TT.txt

Where _ll is the language, and _TT is the country or territory, and where, when taken together, constitiute a locale.

For instance, a Chinese mainland file name would be:
SAPODAAgent_zh_CN.txt.

The same file name for Taiwan would be:
SAPODAAgent_zh_TW.txt.

The Business Object Designer uses this information when selecting a message file. The default search order is to first look for the locale-specific file that matches the locale in which the Business Object Designer is running. If that is not found, the Business Object Designer defaults to the English-US (en_US) version, and finally, the Business Object Designer looks for the file name without any locale or language information.

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 SAPODA1, the tool assumes that the name of the associated message file is SAPODA1Agent.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.

Note: If a non-English locale is required, the same naming convention is still applicable; for example, SAPODA1Agent_zh_TW.txt.

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 297

During the configuration process, you specify:

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

Table 54 describes the tracing level values.

Table 54. 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 SAPODA spawned. • Traces the business object definition dump

For information on where you configure these values, see "Configure initialization properties" on page 297.

Using SAPODA in Business Object Designer

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

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

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

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

   Business Object Designer displays the first window in the wizard, named Select Agent. Figure 34 on page 297 illustrates this window.

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

1. "Select the ODA."
2. "Configure initialization properties" on page 297.
3. "Expand nodes and select objects" on page 299.
4. "Confirm selection of objects" on page 301.
5. "Generate the definition" on page 302 and, optionally, "Provide additional information" on page 303.
6. "Save the definition" on page 310.

Select the ODA

Figure 34 illustrates the first dialog box in Business Object Designer's six-step wizard. From this window, select the ODA to run.

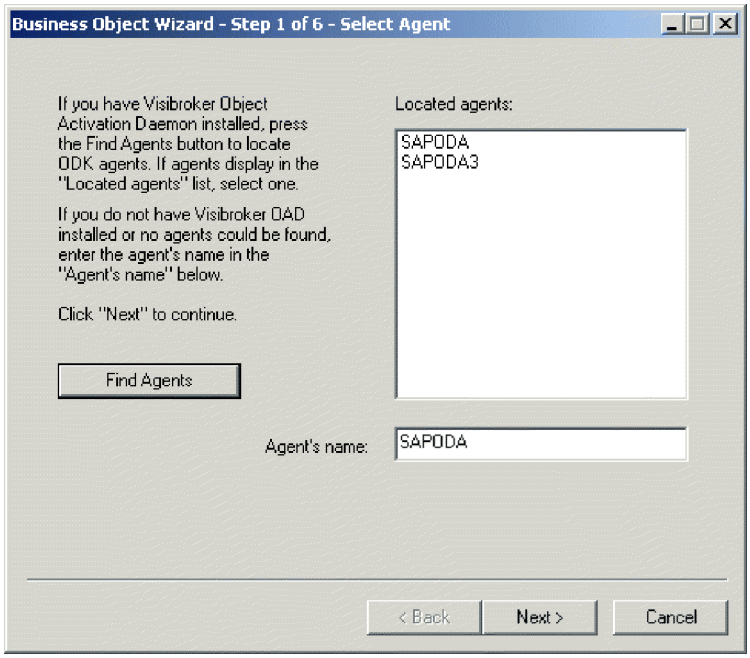


Figure 34. Selecting the ODA

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, enter the Agent's name directly into the Agent's name field. If it still does not locate the 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 SAPODA, it prompts you to enter a set of initialization properties as shown in Figure 35. You can save these properties in a named profile so that you do not need to re-enter them each time you use SAPODA. For information on specifying an ODA profile, see the *Business Object Development Guide*.

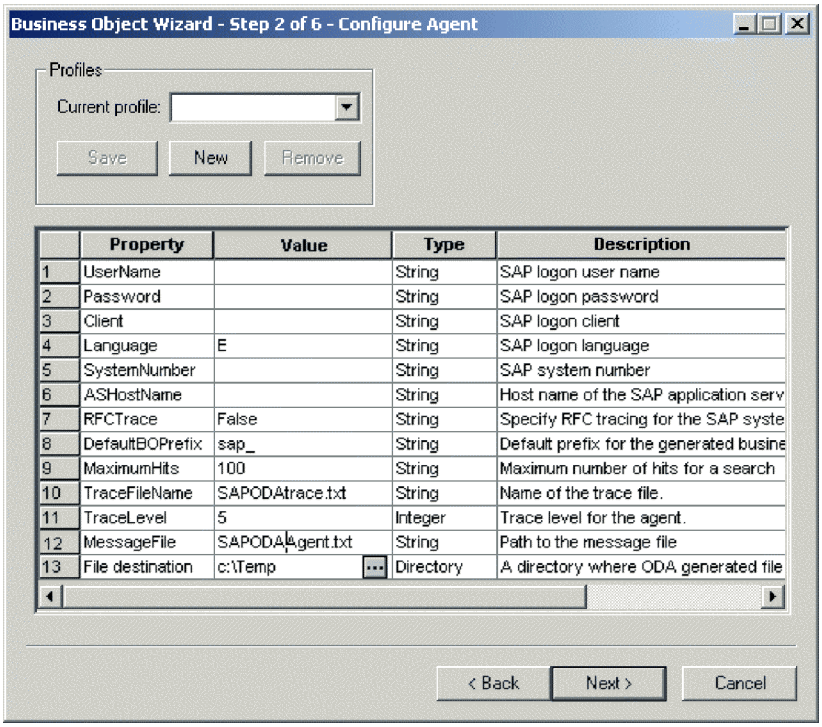


Figure 35. Configuring Agent Properties

Configure the SAPODA properties described in Table 55.

Table 55. SAPODA Properties

Row #	Property Name	Property Type	Description
1	`UserName`	String	SAP logon user name (not required when generating a definition only from an IDoc)
2	`Password`	String	SAP logon password (not required when generating a definition only from an IDoc)
3	`Client`	String	SAP logon client number (not required when generating a definition only from an IDoc)
4	`Language`	String	SAP logon language (not required when generating a definition only from an IDoc)
5	`SystemNumber`	String	SAP system number (not required when generating a definition only from an IDoc)
6	`ASHostName`	String	Host name of the SAP application server (not required when generating a definition only from an IDoc)
7	`RFCTrace`	Single-card, multi-value	RFC tracing for the SAP system
8	`DefaultBOPrefix`	String	Text that is prepended to the name of the business object to make it unique. You can change this later, if required, when Business Object Designer prompts you for Business Object Specific Properties. For more information, see “Provide additional information” on page 303.
9	`MaximumHits`	String	Maximum number of objects returned during a node search. For more information, see “Expand nodes and select objects” on page 299. Default is: `100`

Table 55. SAPODA Properties (continued)

Row #	Property Name	Property Type	Description
10	TraceFileName	String	Name of the trace file. If the file does not exist, SAPODA creates it in the \ODA\SAP directory. If the file already exists, SAPODA appends to it. SAPODA names the file according to the naming convention. For example, if the agent is named SAPODA, it generates a trace file named SAPODAtrace.txt. Use this property to specify a different name for this file.
11	TraceLevel	Integer	Level of tracing enabled for SAPODA For more information on tracing, see "Working with error and trace message files" on page 295.
12	MessageFile	String	Name of the error and message file. SAPODA names the file according to the naming convention. For example, if the agent is named SAPODA, it names the message file SAPODAAgent.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.
13	File destination	Directory	Directory where ODA-generated files are stored. Default is the default directory on the Windows system. It is recommended that you change the default setting to the \connectors\SAP\utilities\generatedfiles directory.

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 objects

After you configure all properties for SAPODA, Business Object Designer displays a tree with the following the initial nodes:

- IDoc types—You can:
  - browse for extracted IDoc definition files
  - select IDocs in the SAP system (Basic IDoc Types and Extension Types)

    Note: Extension Types are customer-defined IDoc Types.
- BOR—select objects that represent BAPIs from the SAP application
- RFC—select objects that represent RFC-enabled functions from the SAP application
- Dynamic Transaction and Retrieve—select the definitions that represent objects from the dynamic transaction and dynamic retrieve metadata tables

– HDR—select the tables required to represent an entity for SAP transactions processed by the Hierarchical Dynamic Retrieve module

The nodes whose names are preceded by a plus sign (+) are expandable. Click on them to display more nodes or leaves. SAPODA generates business object definitions only from leaves.

Figure 36 illustrates this dialog box as originally displayed and with some nodes expanded.

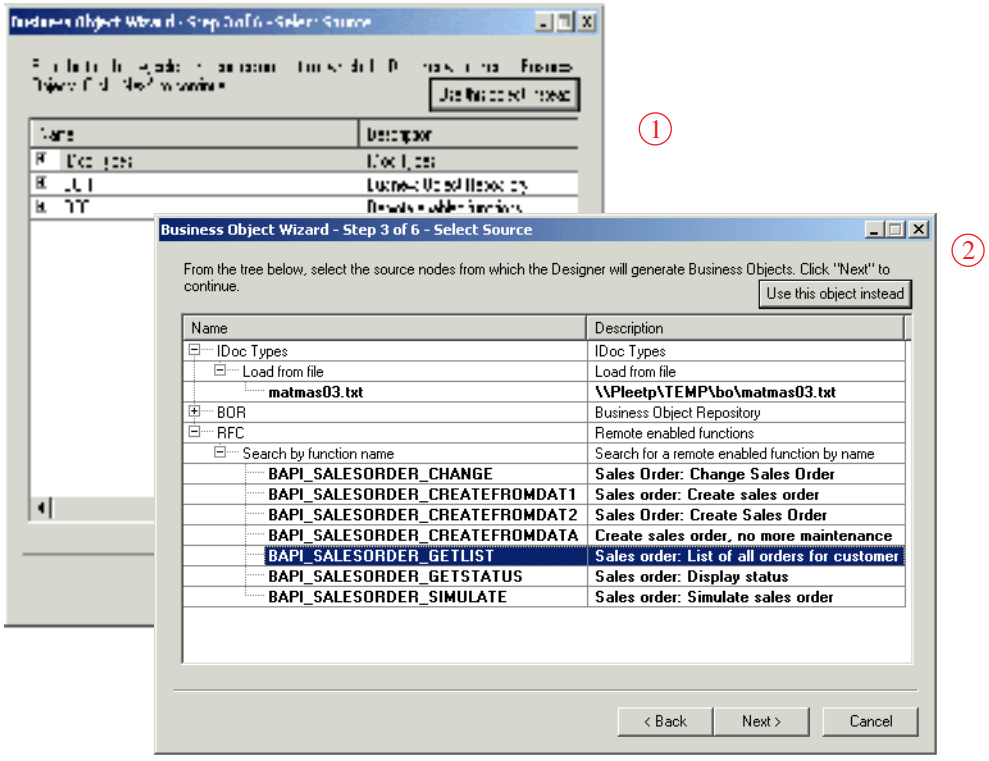


Figure 36. Tree of schema with expanded nodes

When a leaf's name is displayed in bold type, you can select the leaf as the basis for its business object to be generated. Use standard Windows procedures to select multiple leaves. In other words, depress the CTRL key while you use the mouse to select multiple leaves.

Important: On a Windows system, if Business Object Designer cannot find required library files in the Path environment variable or the files are not on the system, it displays a CORBA Exception while attempting to get the tree nodes. For information about these files, see "Before running SAPODA" on page 292.

SAPODA uses a polymorphic node type that allows you to associate a flat file with a node. Initially the node displays without any leaves. You can browse a file system and select files to add to that node. The node is called **polymorphic** because its nature changes from a leaf to a branch when you associate it to one or more files.

Figure 37 illustrates two ways of limiting the number of leaves that Business Object Designer returns:

- A context-sensitive menu that allows you to open a window for browsing files. From this window, you can select which files to associate.
- A wizard that allows you to specify search characters in the objects' names.

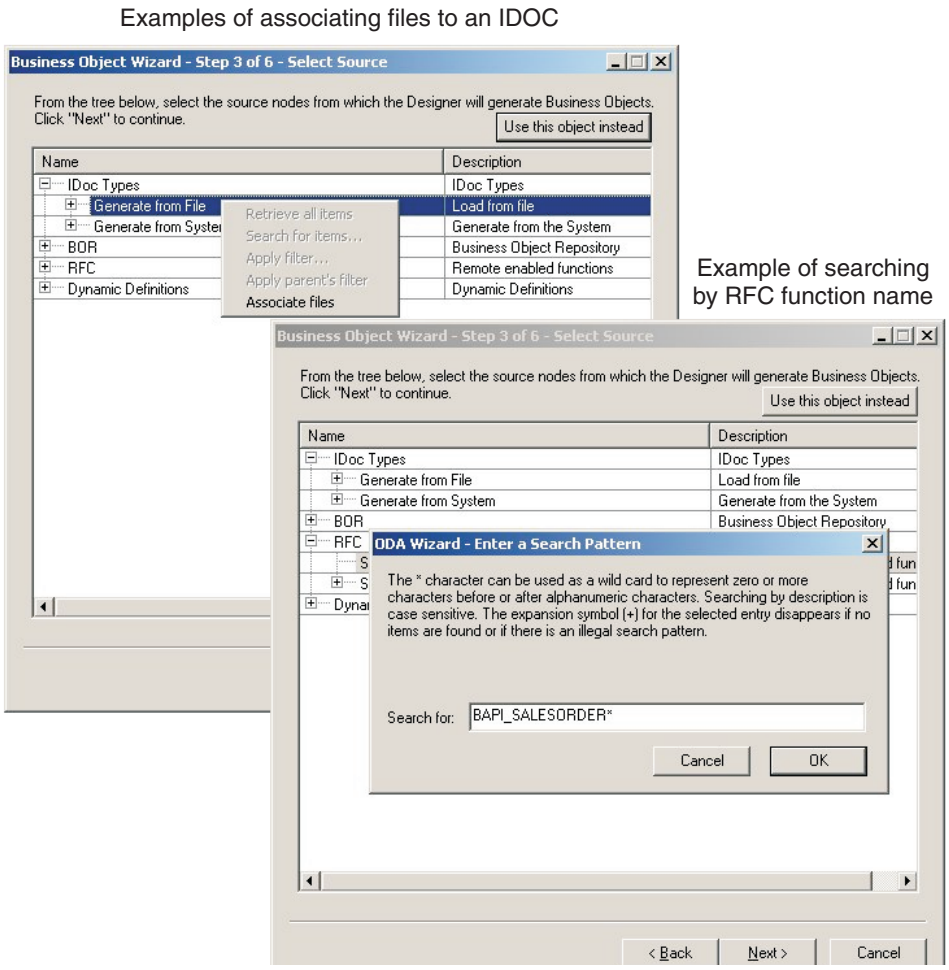


Figure 37. Associating a file and entering search criteria

After you have selected all desired leaves for object generation, click the Next button. For information on how to filter the objects returned, see the *Business Object Development Guide*.

Confirm selection of objects

After you identify all the objects to be associated with a generated business object definition, Business Object Designer displays the dialog box with only the selected leaves and their node paths. Figure 38 illustrates this dialog box.

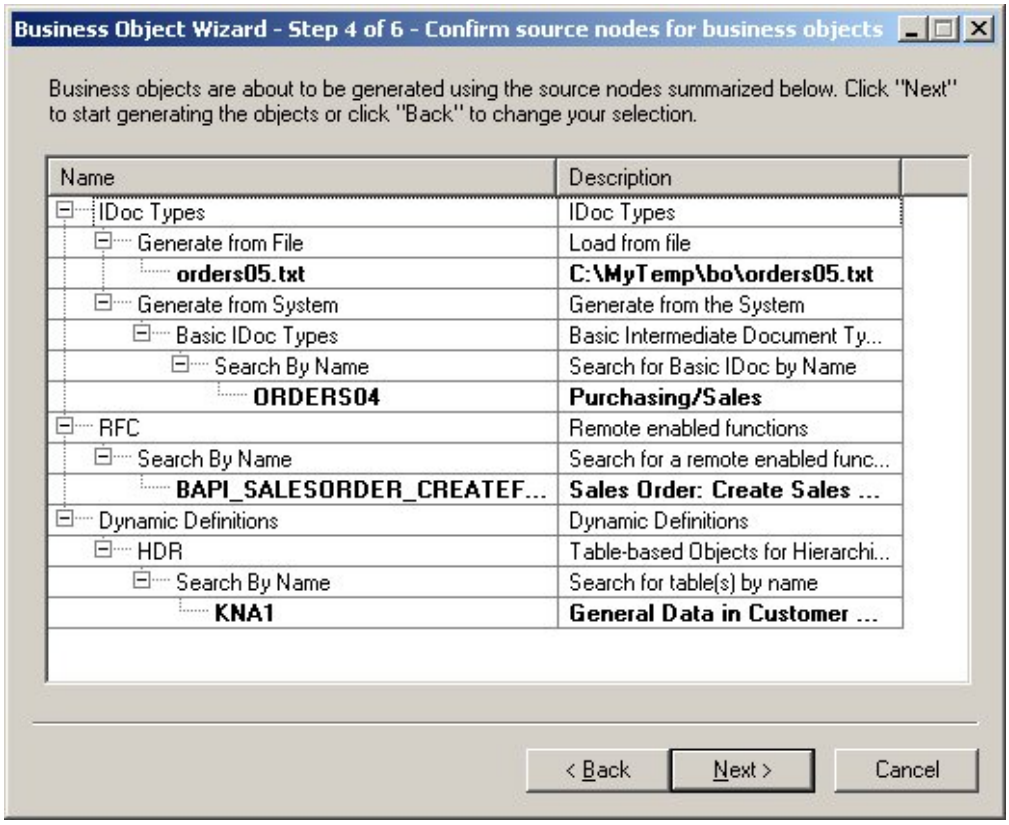


Figure 38. Confirming selection of nodes and leaves

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

Figure 39 illustrates this dialog box.

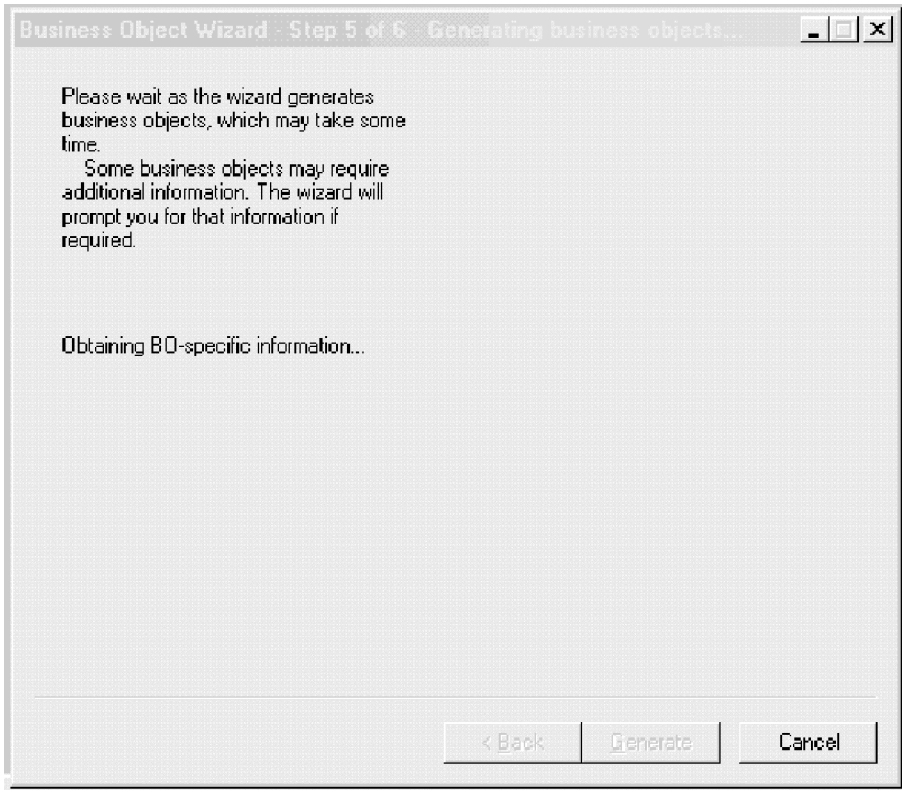


Figure 39. Generating the Definition

Provide additional information

SAPODA prompts for additional information. The type of the top-level node (IDoc types, BOR, RFC, or Dynamic Definitions) determines:

- The set of properties that Business Object Designer displays in the BO Properties window.
- Whether Business Object Designer displays a second window that prompts you for additional object generation information.

This section describes:

- "IDoc Type: Providing additional information"
- "BOR or RFC: Providing additional information" on page 306
- "HDR: Providing additional information" on page 308

Note: No additional information is required for the Dynamic Transaction and Retrieval nodes.

IDoc Type: Providing additional information

SAPODA displays the BO Properties window to enable you to specify information required for business objects based on IDoc types. The properties displayed in this window differ depending on the source of the IDoc (an extracted file or a definition in the SAP system) and whether the definition is being defined for the ABAP Extension module. This section describes the following topics:

- "The BO Properties Window—Common Properties" on page 304

- "The BO Properties Window—Property for IDoc Defined in the SAP System"
- "The BO Properties Window—Specifying a Function module for the ABAP Handler" on page 305

The BO Properties Window—Common Properties: Regardless of whether SAPODA is generating the business object definition from an IDoc file or an IDoc defined in the SAP system, the BO Properties window for an IDoc type allows you to specify or change:

- Prefix information.

  The prefix is text 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 35 on page 298), you do not need to change the value here.

- Module type

  The module type choices are ALE or Extension. Because the ALE and the ABAP Extension modules have different requirements for their business object definitions, it is important to specify which module will be using the business object.

  Note: If there are multiple segments at the top-level of the IDoc, when SAPODA generates the business object definition for the ABAP Extension module, it uses the first IDoc segment to represent the top-level business object. SAPODA represents the other top-level segments as child business objects

The BO Properties Window—Property for IDoc Defined in the SAP System: In addition to the Prefix and module properties, the BO Properties window representing an IDoc defined in the SAP system also displays the Release property. You can use this property to identify an earlier version of the IDoc type.

Important: If the earlier version of the IDoc type has fewer segments than the current version, SAPODA might create a definition with missing segments or SAPODA might display an error indicating that the generation of the business object definition was unsuccessful. This inconsistency is due to different versions of SAP requiring different API calls.

Figure 40 illustrates the two versions of the BO Properties window, one for an extracted IDoc Type definition file and one for an IDoc defined in the SAP system.

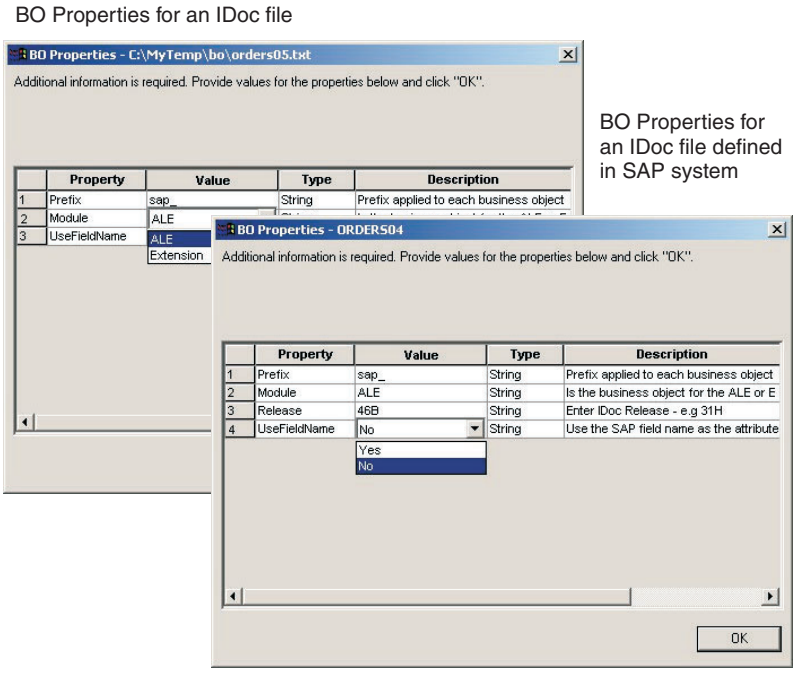


Figure 40. Providing additional information for an IDoc type business object

The BO Properties Window—Specifying a Function module for the ABAP Handler: If you select `Extension` as the module type, SAPODA prompts whether you want to enter function module names for any of the default verbs.

By default, when generating a definition for the ABAP Extension module, SAPODA specifies the following text for verb application-specific information at the business object level of the top-level business object:

```
:/CWLD/IDOC_HANDLER
```

If you already know the function module names to pass to the ABAP handler, select `Yes` at this prompt. SAPODA displays the window illustrated in Figure 41.

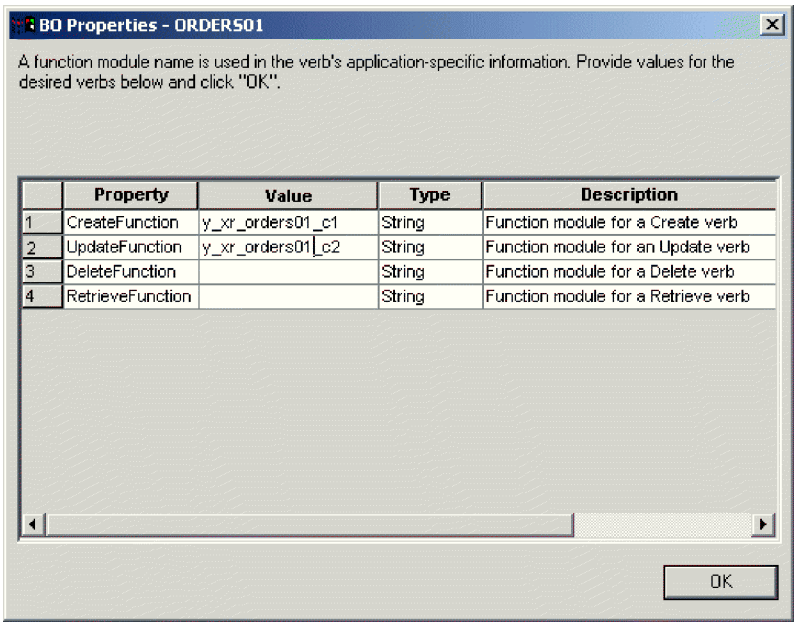


Figure 41. Specifying function modules for the ABAP handler

Figure 41 illustrates a BO Properties window in which two function modules have been specified.

After you save the business object definition, the General tab in Business Object Designer displays the required application-specific information at the business object level of the topmost business object. Figure 42 illustrates such a window with the two specified function modules.

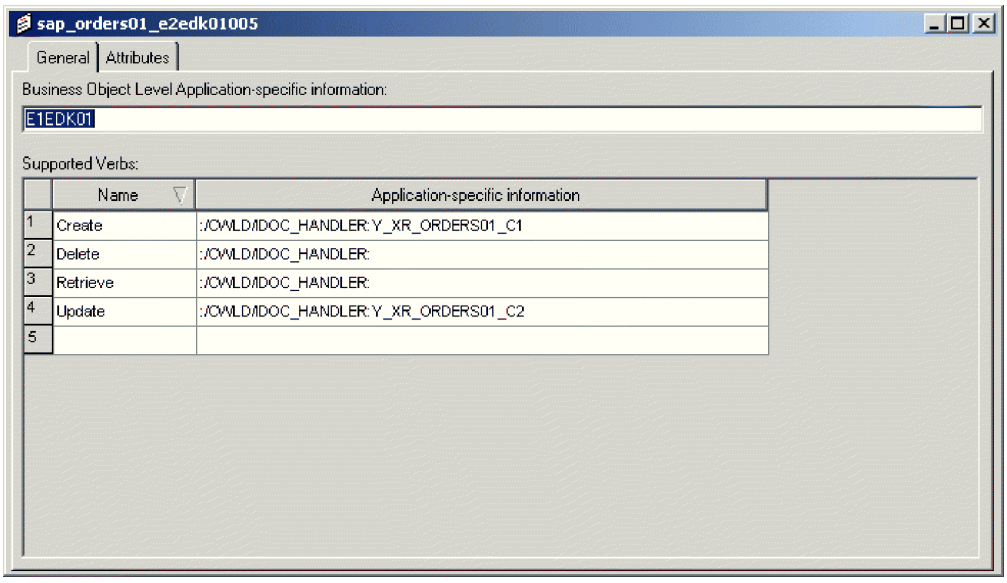


Figure 42. Specifying the ABAP handler in Business Object Designer

BOR or RFC: Providing additional information

There are two BO Properties windows for a BOR or RFC Type. The properties displayed in the first window allow you to specify or change:

- `Prefix` —If you are satisfied with the value you entered for the `DefaultBOPrefix` property in the Configure Agent window (Figure 35 on page 298), you do not need to change the value here.
- `Verb` —Specify the verb.
- `Server Support`—If the definition is to be generated for the connector's RFC Server module, specify `yes`. If the definition is to be generated for the connector's BAPI module, specify `no`.

After you click OK to move forward from the first BO Properties window, SAPODA gives you the opportunity to reduce the size of the generated definition. You are prompted whether you want to remove from the definition any attributes that represent optional parameters. This prompt displays only if there are optional parameters to remove. Reducing the size of the definition can enhance performance later when the connector processes instances of the business object.

Figure 43 illustrates the properties displayed for a BOR or RFC-type object and the prompt that displays after you click OK.

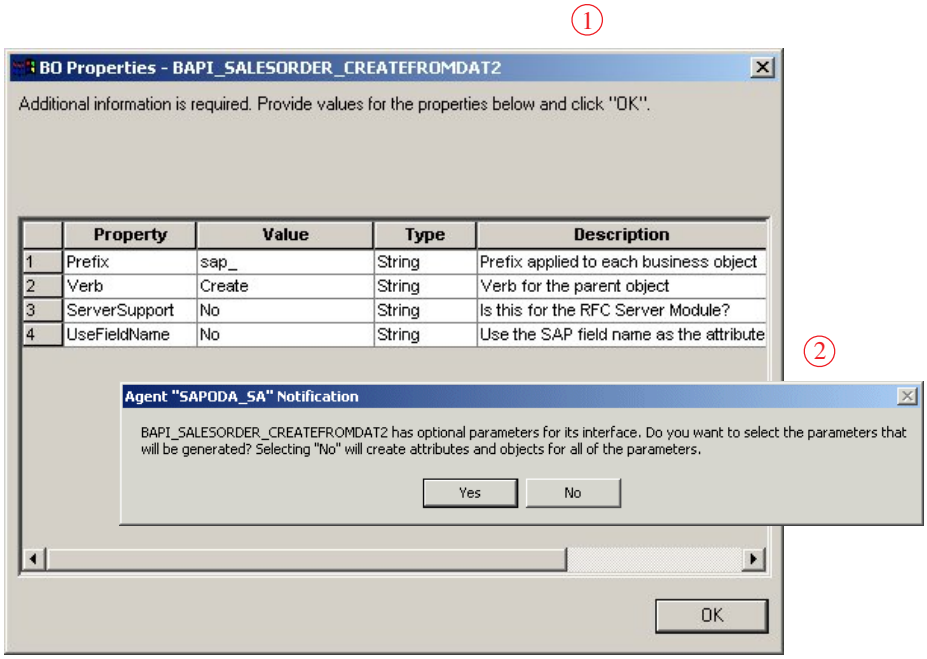


Figure 43. Providing sdditional iInformation for BOR or RFC business objects

If you click Yes in the prompt illustrated above, the second BO Properties window displays. You can specify removal of each optional parameter of a BAPI/RFC interface by changing its Value from `Yes` (include a corresponding attribute in the generated definition) to `No` (do not include an attribute).

If you click No in the prompt illustrated above, the final wizard displays. For more information, see "Save the definition" on page 310.

Figure 44 illustrates the second BO Properties window.

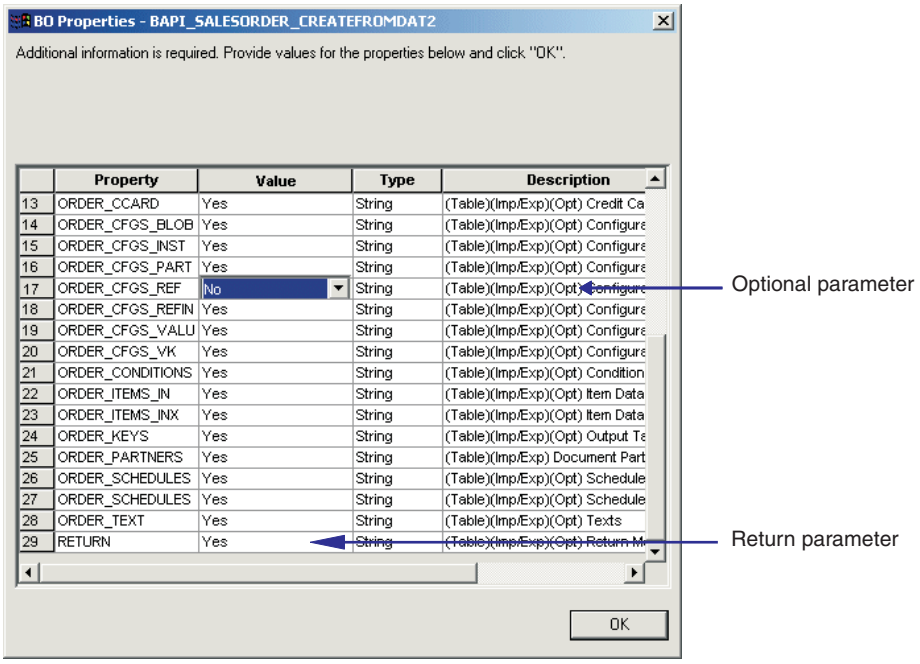


Figure 44. Specifying attributes for removal from the definition

Important: A business object definition for a RFC-enabled function beginning with "`Bapi`" must have an attribute that represents a business object corresponding to a return structure or table. If a definition lacks such an attribute, an error occurs when its corresponding generated code is compiled. If you get this compile error, examine the BAPI to determine if SAP was using a different return structure. In this case, change the generated Java code to point to the proper parameter.

In addition to the specifications you provide in SAPODA, when you create a definition for the RFC Server module, you may want to modify application-specific information after you save the business object definition. For more information, see Chapter 19, "Developing business objects for the RFC Server Module," on page 191.

HDR: Providing additional information

There are two BO Properties window for an HDR table-based object. The property displayed in the first window allows you to specify or change the business object's prefix. If you are satisfied with the value you entered for the DefaultBOPrefix property in the Configure Agent window (Figure 35 on page 298), you do not need to change the value here.

Figure 45 illustrates this window.

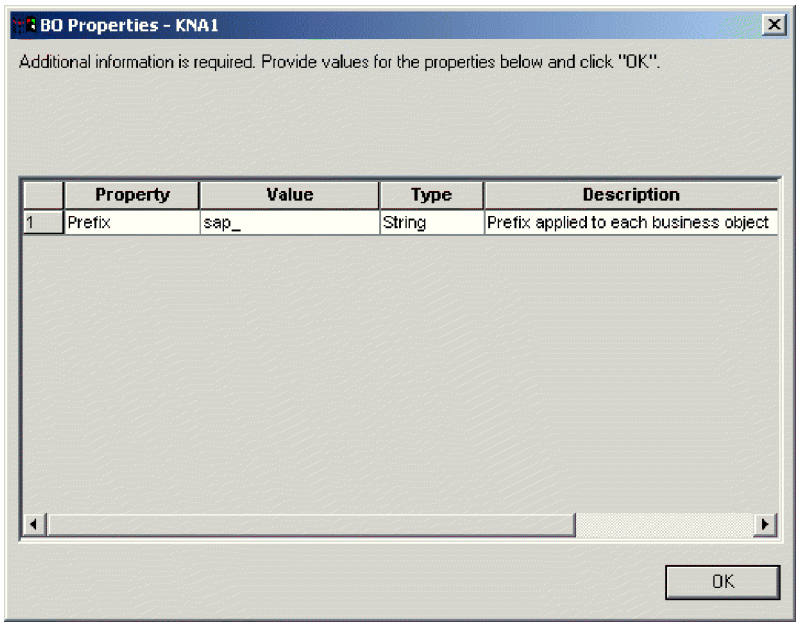


Figure 45. Providing additional information for an HDR business object

In addition, only 512 bytes of information from a table can be returned. When a table returns more than 512 bytes, you will be presented with the dialog found in Figure 46. Answering "No" returns attributes (column descriptions) from the beginning of the table until the 512 byte maximum is reached.

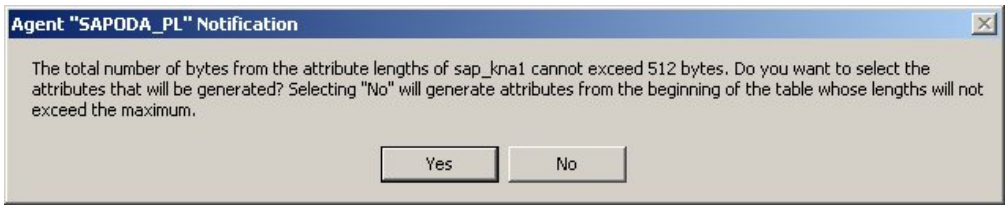


Figure 46. 512 byte warning

Answering "Yes" displays the second BO properties windows noted in Figure 47. The length in bytes for each attribute is provided in the window description. You can specific the inclusion or exclusion of an attribute for the business object by toggling its value between "Yes" and "No."

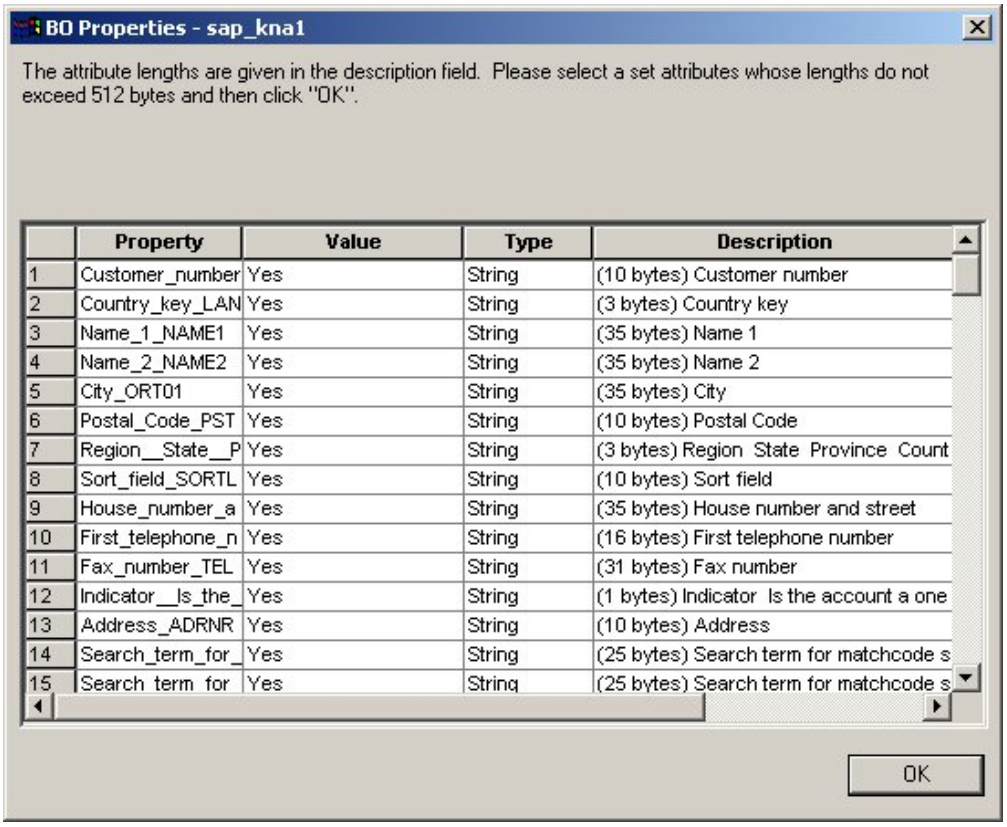


Figure 47. Size and type of BO properties for an HDR business object

For more information, see Chapter 22, "Developing business objects for the Hierarchical Dynamic Retrieve Module," on page 211.

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. Here, you can save the definition to the server or to a file, or you can open the definition for editing in Business Object Designer. For more information, and to make further modifications, see the *Business Object Development Guide*.

Note: When this step completes, Business Object Designer ends a manually-started ODA. To generate another definition, you must restart the ODA.

Figure 48 illustrates this dialog box.

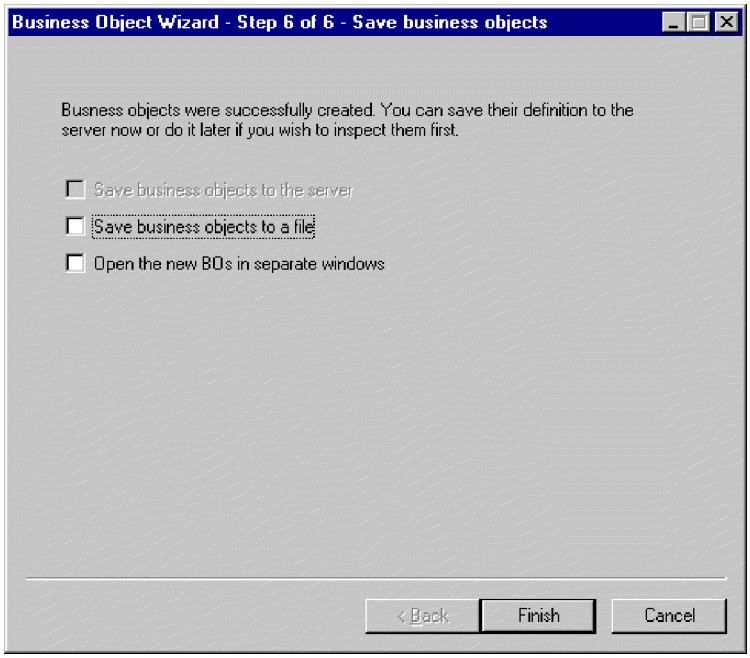


Figure 48. Saving business object definition

After using SAPODA

The business object definition that SAPODA generates from an IDoc Type, BAPI interface, or RFC-enabled function does not contain all the information required for the connector to process the business object. Therefore, after SAPODA finishes generating the definition, you must add all required information to the definition. Use Business Object Designer to examine and modify the business object definition, and to reload or copy a revised definition into the repository.

For information on modifying a business object definition, see the *Business Object Development Guide*. For information on the business object definition that a specific connector module requires, and the modifications you must make before the connector can process it, see the appropriate module's documentation:

- Chapter 13, "Developing business objects for the ALE module," on page 145
- Chapter 16, "Developing business objects for the BAPI Module," on page 169
- Chapter 19, "Developing business objects for the RFC Server Module," on page 191

Index

D

E

F

G

I

Notices

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

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

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

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

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

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

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

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

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

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

Burlingame, CA 94010
U.S.A

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

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

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

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

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

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

Programming interface information

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

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

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

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

Trademarks and service marks

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

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

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

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

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

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

WebSphere Business Integration Express for Item Synchronization V4.3.1,

WebSphere Business Integration Express Plus for Item Synchronization V4.3.1

Printed in USA