ORACLE
JD EDWARDS ENTERPRISEONE

JD Edwards EnterpriseOne Tools 8.96 Connectors Guide

April 2006

ORACLE

JD Edwards EnterpriseOne Tools 8.96 Connectors Guide
SKU E1_TOOLS896TCN-B 0406

Contents

About This Documentation Preface

JD Edwards EnterpriseOne implementation guides provide you with the information that you need to implement and use JD Edwards EnterpriseOne applications from Oracle.

This preface discusses:

- JD Edwards EnterpriseOne application prerequisites.
- Application fundamentals.
- Documentation updates and printed documentation.
- Additional resources.
- Typographical conventions and visual cues.
- Comments and suggestions.
- Common fields in implementation guides.

Note. Implementation guides document only elements, such as fields and check boxes, that require additional explanation. If an element is not documented with the process or task in which it is used, then either it requires no additional explanation or it is documented with common fields for the section, chapter, implementation guide, or product line. Fields that are common to all JD Edwards EnterpriseOne applications are defined in this preface.

JD Edwards EnterpriseOne Application Prerequisites

To benefit fully from the information that is covered in these books, you should have a basic understanding of how to use JD Edwards EnterpriseOne applications.

You might also want to complete at least one introductory training course, if applicable.

You should be familiar with navigating the system and adding, updating, and deleting information by using JD Edwards EnterpriseOne menus, forms, or windows. You should also be comfortable using the World Wide Web and the Microsoft Windows or Windows NT graphical user interface.

These books do not review navigation and other basics. They present the information that you need to use the system and implement your JD Edwards EnterpriseOne applications most effectively.

Application Fundamentals

Each application implementation guide provides implementation and processing information for your JD Edwards EnterpriseOne applications.

For some applications, additional, essential information describing the setup and design of your system appears in a companion volume of documentation called the application fundamentals implementation guide. Most product lines have a version of the application fundamentals implementation guide. The preface of each implementation guide identifies the application fundamentals implementation guides that are associated with that implementation guide.

The application fundamentals implementation guide consists of important topics that apply to many or all JD Edwards EnterpriseOne applications. Whether you are implementing a single application, some combination of applications within the product line, or the entire product line, you should be familiar with the contents of the appropriate application fundamentals implementation guides. They provide the starting points for fundamental implementation tasks.

Documentation Updates and Printed Documentation

This section discusses how to:

- Obtain documentation updates.
- Order printed documentation.

Obtaining Documentation Updates

You can find updates and additional documentation for this release, as well as previous releases, on Oracle's PeopleSoft Customer Connection website. Through the Documentation section of Oracle's PeopleSoft Customer Connection, you can download files to add to your Implementation Guides Library. You'll find a variety of useful and timely materials, including updates to the full line of JD Edwards EnterpriseOne documentation that is delivered on your implementation guides CD-ROM.

Important! Before you upgrade, you must check Oracle's PeopleSoft Customer Connection for updates to the upgrade instructions. Oracle continually posts updates as the upgrade process is refined.

See Also

Oracle's PeopleSoft Customer Connection, http://www.oracle.com/support/support_peoplesoft.html

Ordering Printed Documentation

You can order printed, bound volumes of the complete line of JD Edwards EnterpriseOne documentation that is delivered on your implementation guide CD-ROM. Oracle makes printed documentation available for each major release of JD Edwards EnterpriseOne shortly after the software is shipped. Customers and partners can order this printed documentation by using any of these methods:

- Web
- Telephone
- Email

Web

From the Documentation section of Oracle's PeopleSoft Customer Connection website, access the PeopleBooks Press website under the Ordering PeopleBooks topic. Use a credit card, money order, cashier's check, or purchase order to place your order.

Telephone

Contact MMA Partners, the book print vendor, at 877 588 2525.

Email

Send email to MMA Partners at peoplebookspress@mmapartner.com.

See Also

Oracle's PeopleSoft Customer Connection, http://www.oracle.com/support/support_peoplesoft.html

Additional Resources

The following resources are located on Oracle's PeopleSoft Customer Connection website:

Resource	Navigation
Application maintenance information	Updates + Fixes
Business process diagrams	Support, Documentation, Business Process Maps
Interactive Services Repository	Support, Documentation, Interactive Services Repository
Hardware and software requirements	Implement, Optimize, and Upgrade; Implementation Guide; Implementation Documentation and Software; Hardware and Software Requirements
Installation guides	Implement, Optimize, and Upgrade; Implementation Guide; Implementation Documentation and Software; Installation Guides and Notes
Integration information	Implement, Optimize, and Upgrade; Implementation Guide; Implementation Documentation and Software; Pre-Built Integrations for PeopleSoft Enterprise and JD Edwards EnterpriseOne Applications
Minimum technical requirements (MTRs) (JD Edwards EnterpriseOne only)	Implement, Optimize, and Upgrade; Implementation Guide; Supported Platforms
Documentation updates	Support, Documentation, Documentation Updates
Implementation guides support policy	Support, Support Policy
Prerelease notes	Support, Documentation, Documentation Updates, Category, Release Notes
Product release roadmap	Support, Roadmaps + Schedules
Release notes	Support, Documentation, Documentation Updates, Category, Release Notes
Release value proposition	Support, Documentation, Documentation Updates, Category, Release Value Proposition
Statement of direction	Support, Documentation, Documentation Updates, Category, Statement of Direction

Resource	Navigation
Troubleshooting information	Support, Troubleshooting
Upgrade documentation	Support, Documentation, Upgrade Documentation and Scripts

Typographical Conventions and Visual Cues

This section discusses:

- Typographical conventions.
- Visual cues.
- Country, region, and industry identifiers.
- Currency codes.

Typographical Conventions

This table contains the typographical conventions that are used in implementation guides:

Typographical Convention or Visual Cue	Description
Bold	Indicates PeopleCode function names, business function names, event names, system function names, method names, language constructs, and PeopleCode reserved words that must be included literally in the function call.
Italics	Indicates field values, emphasis, and JD Edwards EnterpriseOne or other book-length publication titles. In PeopleCode syntax, italic items are placeholders for arguments that your program must supply. We also use italics when we refer to words as words or letters as letters, as in the following: Enter the letter *O*.
KEY+KEY	Indicates a key combination action. For example, a plus sign (+) between keys means that you must hold down the first key while you press the second key. For ALT+W, hold down the ALT key while you press the W key.
`Monospace font`	Indicates a PeopleCode program or other code example.
“ ” (quotation marks)	Indicate chapter titles in cross-references and words that are used differently from their intended meanings.

Typographical Convention or Visual Cue	Description
. . . (ellipses)	Indicate that the preceding item or series can be repeated any number of times in PeopleCode syntax.
{ } (curly braces)	Indicate a choice between two options in PeopleCode syntax. Options are separated by a pipe (|).
[ ] (square brackets)	Indicate optional items in PeopleCode syntax.
& (ampersand)	When placed before a parameter in PeopleCode syntax, an ampersand indicates that the parameter is an already instantiated object. Ampersands also precede all PeopleCode variables.

Visual Cues

Implementation guides contain the following visual cues.

Notes

Notes indicate information that you should pay particular attention to as you work with the JD Edwards EnterpriseOne system.

Note. Example of a note.

If the note is preceded by *Important!,* the note is crucial and includes information that concerns what you must do for the system to function properly.

Important! Example of an important note.

Warnings

Warnings indicate crucial configuration considerations. Pay close attention to warning messages.

Warning! Example of a warning.

Cross-References

Implementation guides provide cross-references either under the heading "See Also" or on a separate line preceded by the word *See.* Cross-references lead to other documentation that is pertinent to the immediately preceding documentation.

Country, Region, and Industry Identifiers

Information that applies only to a specific country, region, or industry is preceded by a standard identifier in parentheses. This identifier typically appears at the beginning of a section heading, but it may also appear at the beginning of a note or other text.

Example of a country-specific heading: “(FRA) Hiring an Employee”

Example of a region-specific heading: “(Latin America) Setting Up Depreciation”

Country Identifiers

Countries are identified with the International Organization for Standardization (ISO) country code.

Region Identifiers

Regions are identified by the region name. The following region identifiers may appear in implementation guides:

- Asia Pacific
- Europe
- Latin America
- North America

Industry Identifiers

Industries are identified by the industry name or by an abbreviation for that industry. The following industry identifiers may appear in implementation guides:

- USF (U.S. Federal)
- E&G (Education and Government)

Currency Codes

Monetary amounts are identified by the ISO currency code.

Comments and Suggestions

Your comments are important to us. We encourage you to tell us what you like, or what you would like to see changed about implementation guides and other Oracle reference and training materials. Please send your suggestions to Documentation Manager, Oracle Corporation, 7604 Technology Way, Denver, CO, 80237. Or email us at documentation_us@oracle.com.

While we cannot guarantee to answer every email message, we will pay careful attention to your comments and suggestions.

Common Fields Used in Implementation Guides

Address Book Number Enter a unique number that identifies the master record for the entity. An address book number can be the identifier for a customer, supplier, company, employee, applicant, participant, tenant, location, and so on. Depending on the application, the field on the form might refer to the address book number as the customer number, supplier number, or company number, employee or applicant ID, participant number, and so on.

As If Currency Code Enter the three-character code to specify the currency that you want to use to view transaction amounts. This code enables you to view the transaction amounts as if they were entered in the specified currency rather than the foreign or domestic currency that was used when the transaction was originally entered.

Batch Number Displays a number that identifies a group of transactions to be processed by the system. On entry forms, you can assign the batch number or the system can assign it through the Next Numbers program (P0002).

Batch Date Enter the date in which a batch is created. If you leave this field blank, the system supplies the system date as the batch date.

Batch Status Displays a code from user-defined code (UDC) table 98/IC that indicates the posting status of a batch. Values are:

Blank: Batch is unposted and pending approval.

A: The batch is approved for posting, has no errors and is in balance, but has not yet been posted.

D: The batch posted successfully.

E: The batch is in error. You must correct the batch before it can post.

P: The system is in the process of posting the batch. The batch is unavailable until the posting process is complete. If errors occur during the post, the batch status changes to *E*.

U: The batch is temporarily unavailable because someone is working with it, or the batch appears to be in use because a power failure occurred while the batch was open.

Branch/Plant Enter a code that identifies a separate entity as a warehouse location, job, project, work center, branch, or plant in which distribution and manufacturing activities occur. In some systems, this is called a business unit.

Business Unit Enter the alphanumeric code that identifies a separate entity within a business for which you want to track costs. In some systems, this is called a branch/plant.

Category Code Enter the code that represents a specific category code. Category codes are user-defined codes that you customize to handle the tracking and reporting requirements of your organization.

Company Enter a code that identifies a specific organization, fund, or other reporting entity. The company code must already exist in the F0010 table and must identify a reporting entity that has a complete balance sheet.

Currency Code Enter the three-character code that represents the currency of the transaction. JD Edwards EnterpriseOne provides currency codes that are recognized by the International Organization for Standardization (ISO). The system stores currency codes in the F0013 table.

Document Company Enter the company number associated with the document. This number, used in conjunction with the document number, document type, and general ledger date, uniquely identifies an original document.

If you assign next numbers by company and fiscal year, the system uses the document company to retrieve the correct next number for that company.

If two or more original documents have the same document number and document type, you can use the document company to display the document that you want.

Document Number — Displays a number that identifies the original document, which can be a voucher, invoice, journal entry, or time sheet, and so on. On entry forms, you can assign the original document number or the system can assign it through the Next Numbers program.

Document Type — Enter the two-character UDC, from UDC table 00/DT, that identifies the origin and purpose of the transaction, such as a voucher, invoice, journal entry, or time sheet. JD Edwards EnterpriseOne reserves these prefixes for the document types indicated:

P: Accounts payable documents.

R: Accounts receivable documents.

T: Time and pay documents.

I: Inventory documents.

O: Purchase order documents.

S: Sales order documents.

Effective Date — Enter the date on which an address, item, transaction, or record becomes active. The meaning of this field differs, depending on the program. For example, the effective date can represent any of these dates:

- The date on which a change of address becomes effective.
- The date on which a lease becomes effective.
- The date on which a price becomes effective.
- The date on which the currency exchange rate becomes effective.
- The date on which a tax rate becomes effective.

Fiscal Period and **Fiscal Year** — Enter a number that identifies the general ledger period and year. For many programs, you can leave these fields blank to use the current fiscal period and year defined in the Company Names & Number program (P0010).

G/L Date (general ledger date) — Enter the date that identifies the financial period to which a transaction will be posted. The system compares the date that you enter on the transaction to the fiscal date pattern assigned to the company to retrieve the appropriate fiscal period number and year, as well as to perform date validations.

JD Edwards EnterpriseOne Tools Connectors Preface

This preface discusses Connectors companion documentation.

Connectors Companion Documentation

Additional, essential information describing the setup and design of JD Edwards EnterpriseOne Tools resides in companion documentation. The companion documentation consists of important topics that apply to Connectors as well as other JD Edwards EnterpriseOne Tools. you should be familiar with the contents of these companion guides:

- Interoperability
- Configurable Network Computing Implementation

See Also

JD Edwards EnterpriseOne Tools 8.96 Interoperability Guide, "Getting Started with JD Edwards EnterpriseOne Tools Interoperability"

JD Edwards EnterpriseOne Tools 8.96 Configurable Network Computing Implementation Guide, "Getting Started with JD Edwards EnterpriseOne Tools Configurable Network Computing Implementation," Configurable Network Computing Overview

CHAPTER 1

Getting Started with JD Edwards EnterpriseOne Tools Connectors

This chapter discusses:

- Connectors Overview
- Connectors Implementation

JD Edwards EnterpriseOne Tools Connectors Overview

Connectors are point-to-point component-based interoperability models that enable third-party applications and JD Edwards EnterpriseOne to share logic and data. Oracle's JD Edwards EnterpriseOne connector architecture includes Java and Component Object Model (COM) connectors and provides:

- Access to business functions
- Session management
- Point of entry
- Connection pooling
- Inbound transaction functionality
- Outbound event functionality

Using connectors provides additional benefits, such as:

- Connectors are scalable
- Connectors provide multi-threading
- Connectors enable concurrent users

Oracle's JD Edwards EnterpriseOne supports the COM connector, a Java connector, and a dynamic Java connector. The COM connector is fully compliant with the Microsoft Component Object Model. You can easily tie JD Edwards EnterpriseOne functionality to Visual Basic and VC++ applications. The Java connector is a portable language, so you can easily tie JD Edwards EnterpriseOne functionality to Java applications. The dynamic Java connector provides the same type of functionality as the Java connector but does not require you to generate business functions.

The JD Edwards EnterpriseOne connectors can receive and send XML documents. The connector architecture provides the capability to expose C and Java APIs for XML documents. Some of the benefits of using XML documents are:

- You can use XML documents to aggregate business function calls into one object, which reduces network traffic.

- Because XML processing is based on the connector architecture, XML processing is scalable and multiple connections can be opened.
- XML processing supports XML CallObject, XMLList, and XMLTrans.

Choosing the Connector Solution

Use this list as a guideline to decide which connector is best for you:

- Identify the logic or data that you want to access in JD Edwards EnterpriseOne.
- Decide whether you want to use business functions exposed through a connector directly or XML documents.

Then decide whether to use a COM connector or a Java connector. If you are using an application server, these guidelines can help you decide which connector to select:

- If you are using Site Server, Commerce Server, or .NET, consider the COM connector.
- If you are using a J2EE-based application server, consider the Java connector.
- The Java connector supports Java Connector Architecture Resource Adapter (JCA).

Connectors Implementation

This section provides an overview of the steps that are required to implement a JD Edwards EnterpriseOne Connector.

In the planning phase of your implementation, take advantage of all JD Edwards sources of information, including the installation guides and troubleshooting information. A complete list of these resources appears in the preface in *About This Documentation* with information about where to find the most current version of each.

Connectors Implementation Steps

This table lists the steps for the JD Edwards EnterpriseOne Tools Connectors implementation. After you determine which connector you should use, you must install and configure the connector. Installation and configuration information for COM, dynamic Java, and Java connectors is provided in this document. In addition to the JD Edwards EnterpriseOne Tools and Applications installation guides, you may need to install and configure other servers and systems such as Transaction Server components, the HTML server, and so on. The JD Edwards EnterpriseOne installation guides referenced in this document are available on Customer Connection.

Step	Reference
1. Install JD Edwards EnterpriseOne Tools and set up a user account.	*JD Edwards EnterpriseOne Tools Release 8.96 Installation Guide* on Customer Connection
2. Install JD Edwards EnterpriseOne applications.	*JD Edwards EnterpriseOne Application Release 8.12 Installation Guide* on Customer Connection

CHAPTER 2

Understanding COM Interoperability

This chapter provides an overview of Component Object Model (COM) interoperability and JD Edwards EnterpriseOne COM Interoperability.

COM Interoperability

COM enables developers to build systems by assembling reusable components from different vendors. COM provides logic and data sharing among disparate applications. COM is a binary interoperability specification and communication convention for software components. It is a single-vendor technology that is available on Microsoft platforms only. Since most independent software components are also self-contained, they are frequently called objects or servers.

Being a binary specification, COM is inherently independent of programming languages. Unlike software libraries or DLLs, which are compiled to specific language or linkage conventions, COM-based software components are created ready to work with any COM client. For example, a Visual C++ application can use COM objects created in Visual Basic, or a VBScript within an intranet web page to control a COM object written in MicroFocus COBOL.

The COM connector provides these two types of services on the JD Edwards EnterpriseOne server:

- Business function execution.

  These chapters discuss business function execution:

  - Understanding JD Edwards EnterpriseOne COM Server.
  - Deploying the COM Server for Business Functions.
  - Using COM Transactions.

- Asynchronous event notifications and introspection operations.

  These chapters discuss event notifications and introspection operations:

  - Using COM Connector Events - Classic Events
  - Using COM Connector Events - Guaranteed Events

The COM connector provides a mechanism for executing business functions on the JD Edwards EnterpriseOne server. You use the GenCOM utility on the Microsoft Windows client to generate wrappers for business function objects. The wrappers can be deployed on any machine. You can develop application code for the generated wrappers using Visual Basic (VB) or C++. Once the objects change in the package, the connector communicates with the JD Edwards EnterpriseOne server for login, logoff, transactions, and for each business function execution call. Distributed Component Object Model (DCOM) enables COM objects in a distributed environment. COM+ transactions enables COM applications and third-party applications to take part in distributed transactions.

The COM connector supports subscribe and publish functionality for JD Edwards EnterpriseOne events. These software releases are supported by the COM connector:

- JD Edwards EnterpriseOne Tools 8.96
- JD Edwards EnterpriseOne Tools 8.95
- JD Edwards EnterpriseOne Tools 8.94
- JD Edwards EnterpriseOne 8.93
- JD Edwards EnterpriseOne 8.9

Note. Business function execution is the same in all these releases.

JD Edwards EnterpriseOne COM Interoperability

This section provides an overview about JD Edwards EnterpriseOne COM interoperability and discusses:

- COM objects
- COM interoperability usage

COM Objects

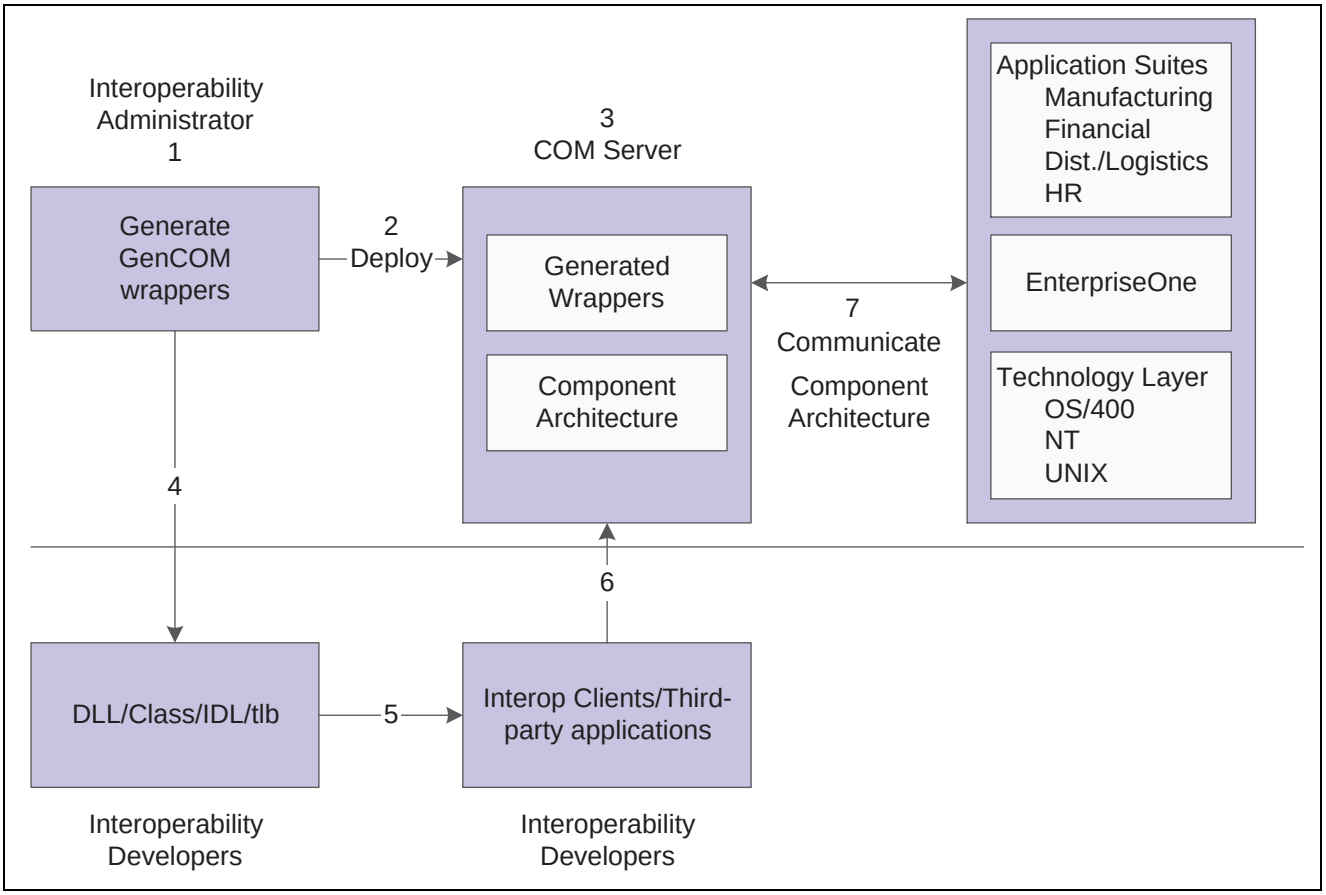
Using COM, JD Edwards EnterpriseOne exposes all master and major business functions through the interface definition language (IDL) standard. A business function is a logical collection of C functions and their associated data structures grouped together to produce a unit of work. With COM, JD Edwards EnterpriseOne can pass logic and data requests to other applications using COM wrappers. COM objects are wrappers around these business functions and data structures. These wrappers provide common interoperability methods across dissimilar systems. A wrapper is attached to each master and major business function and provides stubs for third-party applications to access.

The interface provided by the COM wrappers has a one-to-one correspondence with the business functions. For example, if within the system library a business function named B550001 exists, and within this business function two C functions, named foo1 and foo2 exist with data structures for each function, named DS1 and DS2, the corresponding COM object would be:

```
Interface IDS1
{
}
Interface IDS2
{
}
 Interface IB550001
{
  HRESULT foo1 {IDS1 * param, IConnector* conn, long accessNumber);
    HRESULT foo2 (IDS2 * param, IConnector* conn, long accessNumber);
}
Their associated program IDs (ProgID) would be:
IDS1  - DS1.jdeDS1.1
IDS2  - DS2.jdeDS2.1
IB550001 - B550001.jdeB550001.1
```


COM Interoperability Usage

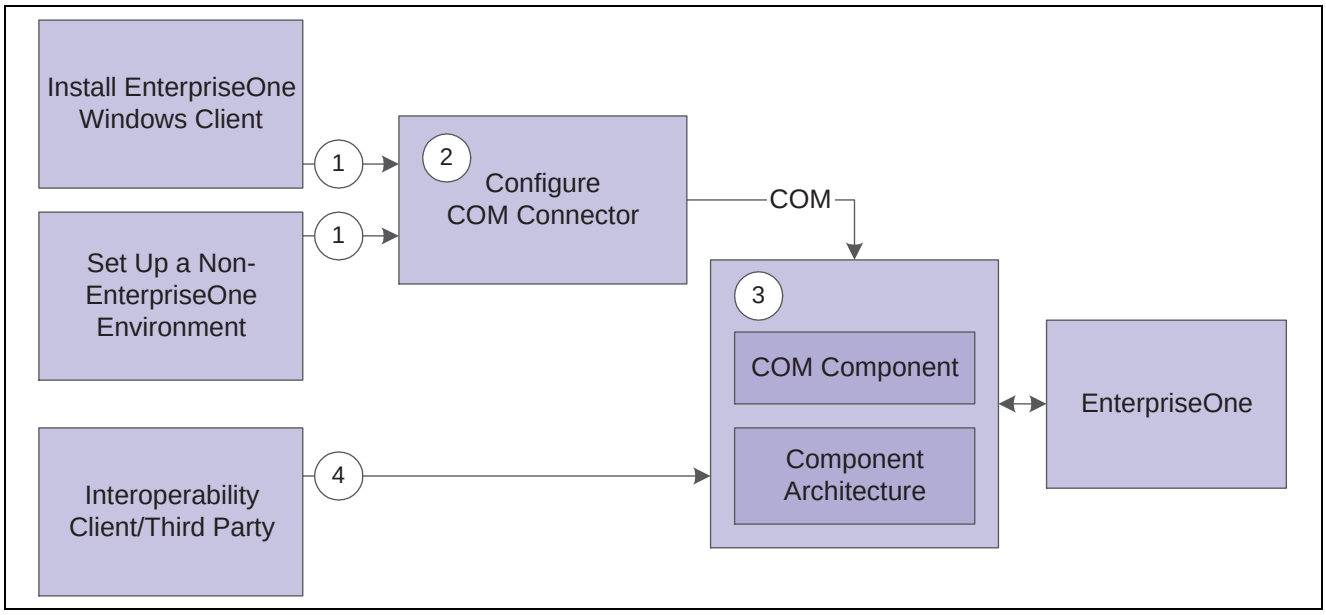
This illustration shows how the COM interoperability solution for business function execution typically flows:



COM interoperability solution for business function execution

1. The administrator generates the COM wrappers.
2. The administrator deploys the COM objects to the COM server.
3. The COM server enables communication with the application server so that the generated COM objects can be used in applications.
4. The COM objects are configured to communicate with the application server once the COM objects are on the COM server.
5. The DLLs or IDLs from the generated COM objects are copied so that developers can use them.
6. The application developers create the applications.
7. The applications communicate with the COM server.

This illustration shows how the COM interoperability solution for event notification and introspection typically flows:



COM interoperability solution — event notification and introspection

1. Install a JD Edwards EnterpriseOne Client.
2. Configure the COM connector.
3. COM Connector enables communications with JD Edwards EnterpriseOne so that clients can introspect and subscribe to events
4. Applications developer crates applications to subscribe to and receive events.

CHAPTER 3

Understanding the COM Solution for Business Function Execution

This chapter discusses:

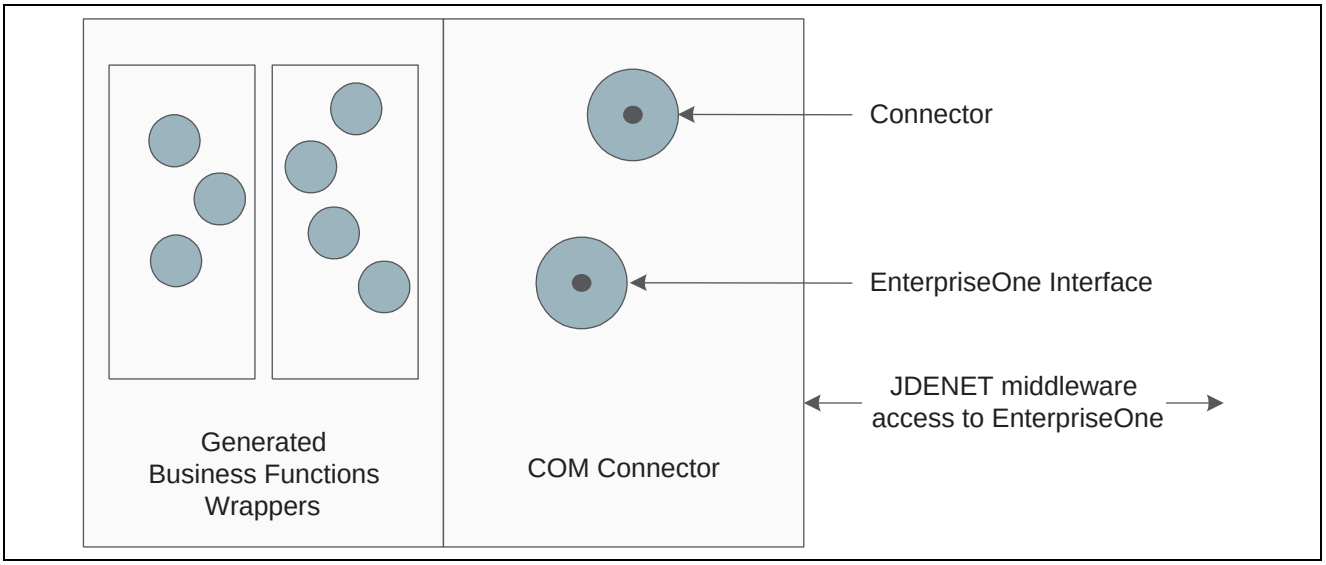
- JD Edwards EnterpriseOne COM server
- COM connector
- Generated COM (GenCOM) components
- COM Wrapper Version Checker (CheckVer)

JD Edwards EnterpriseOne COM Server

The JD Edwards EnterpriseOne COM server contains two parts:

- COM connector.
- Generated JD Edwards EnterpriseOne COM components (wrappers).

This diagram shows the two parts of the COM server:



Parts of the COM server

COM Connector

The COM server provides an interface to JD Edwards EnterpriseOne, executes business functions within valid transactions, and provides error processing for interoperability clients. The main component of the COM server is the COM connector. The COM connector provides COM components that interface with JD Edwards EnterpriseOne and hosts the business component DLL generated by the GenCOM tool. The COM connector also provides the connector component that enables an interoperability client to log in and log out from JD Edwards EnterpriseOne. It manages all user sessions connected to the COM server. This table identifies the binaries that combine to comprise the COM connector:

Binary	Explanation
JDECOMConnector2.exe	Primary interface for login and createBusinessObjects. Also maintains the created users and business objects.
JDECOMMN.dll	Interface for JDEMathNumeric and JDETimeZone.
Callobject.dll	Internal to JDECOMConnector.exe.
Comlog.dll	Used for logging, cache, and OCM lookup.
EventClass.dll	JD Edwards EnterpriseOne event class that is implemented to receive events.
EventListener.dll	Receives events from the JD Edwards EnterpriseOne server and publishes the events to COM+ Events.
EventManager.dll	Provides the interface for subscribe, unsubscribe, getList, and getTemplate for events.
jdeunicode.dll	The Unicode library, which is internal to JD Edwards EnterpriseOne.
OneWorldInterfaceTx.dll	Provides the interface for JD Edwards EnterpriseOne transactions and COM+ two-phase commit transactions.
Xmlinterop.dll	Contains the JDENET transport mechanism and the XMLRequest.
ClientService.dll	Enables event notification and introspection using XML over HTTP protocol. Applicable for JD Edwards EnterpriseOne 8.95 and later Tools releases only.
EventHandler.dll	Receives events from the Transaction server and publishes events to COM+. Applicable for JD Edwards EnterpriseOne 8.95 and later Tools releases only.
JMSEvent.dll	Receives JMS events from the Transaction server and publishes events to COM+. Applicable for JD Edwards EnterpriseOne 8.94 only.

The JDECOMConnector2.idl defines the COM interfaces of the COM connector. JDECOMConnector2.idl is available under the Include directory.

The COM connector is available with the JD Edwards EnterpriseOne server and client install.

GenCOM Components

This section provides an overview of GenCOM and discusses:

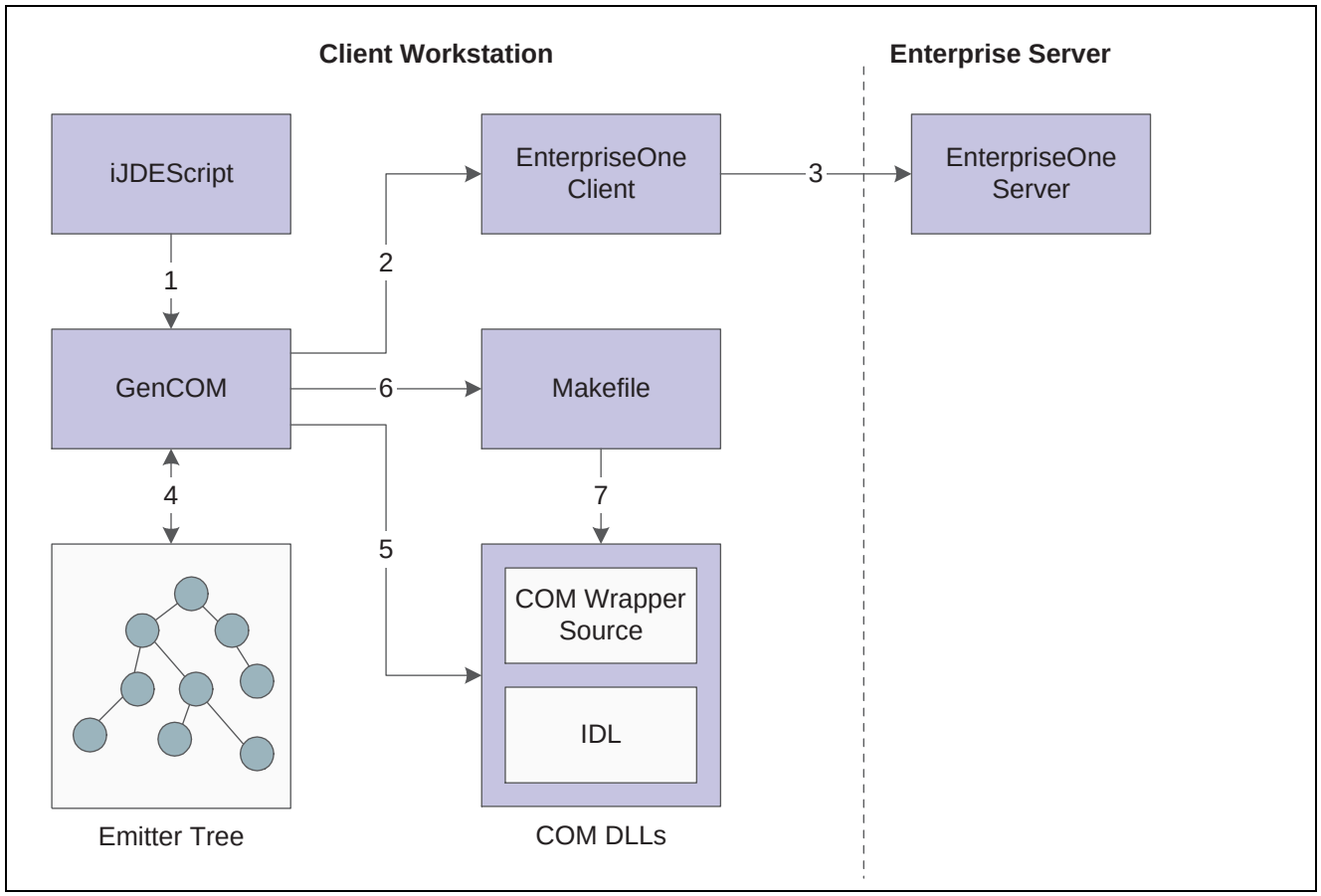
- Installation information.
- ProgID.
- Setting up an environment for GenCOM.
- Running GenCOM.
- Using GenCOM output.

Understanding GenCOM

GenCOM is a client tool that uses a multipass process to generate JD Edwards EnterpriseOne COM components. GenCOM is included in the client installation. The COM Generation Tool is in <install>\system\bin32\GenCOM.exe.

GenCOM is a command line tool that reads a script file to determine which components to generate. GenCOM uses an iJDEScript file as input to generate a COM DLL that is hosted by the COM connector. The iJDEScript file specifies wrapper components for business functions. Once the generated wrapper components are registered to the COM environment, they can be used to access business function functionality.

This illustration shows the process:



GenCOM process

1. GenCOM reads the iJDEScript file.
2. GenCOM retrieves the metadata for the business functions specified in the iJDEScript file.
3. GenCOM resolves dependency on the data structure.
4. GenCOM creates an internal emitter tree for the library to be generated.
5. GenCOM reads each node of the internal emitter tree and generates the appropriate COM code.
6. GenCOM generates a make file.
7. GenCOM compiles and builds the COM DLL from the generated code.

See Chapter 15, "Understanding iJDEScript," page 173.

Installation Information

Because the GenCOM application produces interfaces based on the package currently installed on the machine, installation plans must be made on a site-by-site basis. The DLLs produced are business function release-dependent and can be installed only on machines with the identical packages available.

The GenCOM output is COM servers in the form of DLLs. You can use these DLLs to create an interface with the JD Edwards EnterpriseOne system. You should not assume that a client has installed these servers as part of the standard JD Edwards EnterpriseOne installation. You should provide a full installation of any of the servers the applications require.

ProgID

Each time GenCOM generates a wrapper, it creates a ProgID for each COM component. The ProgID identifies the COM component in the registry. The ProgID is independent of JD Edwards EnterpriseOne and is based on the library and the interface specifications in the script file. The key, OneWorldRelease, contains the JD Edwards EnterpriseOne release and environment information. For example, if the library name is AddressBook and the interface name is JDEAddressBook, then the ProgID will be AddressBook.JDE AddressBook. If GenCOM is run with environment DV9NIS2, then the OneWorldRelease key contains DV9NIS2. If a type mismatch exists, you receive a warning.

The CompatibleEnvironment key remembers the list of JD Edwards EnterpriseOne environments with which the wrapper is compatible. If an environment is not on the list or is listed as incompatible, the COM client receives an error message when trying to create the object with the environment.

This sample code illustrates the standard ProgID naming conventions:

```
HKEY CLASSES ROOT\
CLSID\{77454442-7941-44BB-9BCB-4253E80AC8B3)}
\InprocServer32 C:\B9\System\IDA\Samples\AddressBook\AddressBook.dll
\ProgID AddressBook.JDEAddressBook
\VersionIndependentProgID AddressBook.JDEAddressBook
\OneWorldRelease DV9NIS2
\CompatibleEnvironment DV9NIS2
```

Setting Up an Environment for GenCom

You can use one of these platforms:

- Microsoft Studio 6.0
- Microsoft .NET

Setting Up an Environment for GenCOM on Microsoft Visual Studio 6.0

Setting up a Microsoft Windows NT client environment involves several steps. You should make sure that these items are set up appropriately:

- Include directories
- Lib directories
- MSDev directories
- Paths

Note. Set up this environment for GenGOM if you are using JD Edwards EnterpriseOne Tools 8.94 or an earlier release of JD Edwards EnterpriseOne Tools.

Example: Include Directories

< Directory where Microsoft SDK files are located>\include

Example: C:\ Program Files\Microsoft SDK\include

< Directory where Microsoft program files are located>\VC98\atl\include

Example: C:\ Program Files\Microsoft Visual Studio\VC98\atl\include

< Directory where Microsoft program files are located>\VC98\mfc\include

Example: C:\Program Files\Microsoft Visual Studio\VC98\mfc\include

< Directory where Microsoft program files are located>\VC98\include

Example: C:\Program Files\Microsoft Visual Studio\VC98\include

< Directory where JD Edwards EnterpriseOne is located and release either Master, Prod, or Pristine>\include

Example 1: D:\B9\MSTB9\include

Example 2: D:\B9\PROD\include

< Directory where JD Edwards EnterpriseOne is located and release either Master, Prod, or Pristine>\includeV

Example: D:\B9\SYSTEM\includeV

< Directory where JD Edwards EnterpriseOne is located and release either Master, Prod, or Pristine>\include

Example: D:\B9\SYSTEM\include

Example: Lib Directories

< Directory where Microsoft SDK files are located>\lib

Example: C:\ Program Files\Microsoft SDK\lib

< Directory where Microsoft program files are located >\VC98\mfc\lib

Example: C:\Program Files\Microsoft Visual Studio\VC98\mfc\lib

< Directory where Microsoft program files are located >\VC98\lib

Example: C:\Program Files\Microsoft Visual Studio\VC98\lib

< Directory where Microsoft program files are located >\Common\MSDev98\Bin

Example: C:\Program Files\Microsoft Visual Studio\Common\MSDev98\Bin

< Directory where JD Edwards EnterpriseOne is located>\System\Lib32

Example: D:\B9\System\Lib32

Example: MSDev Directories

< Directory where Microsoft program files are located >\Common\MSDev98

Example: C:\Program Files\Microsoft Visual Studio\Common\MSDev98

< Directory where Microsoft DevStudio is located>\SharedIDE

Example: C:\Program Files\DevStudio\SharedIDE

Example: Paths

< Directory where Microsoft SDK files are located>\bin

Example: C:\ Program Files\Microsoft SDK\bin

< Directory where Windows NT is located>\System32

Example: C:\Winnt\System32

< Directory where Microsoft program files are located >\Common\Tools\Winnt

Example: C:\Program Files\Microsoft Visual Studio\Common\Tools\Winnt

< Directory where Microsoft program files are located >\Common\Msdev98\Bin

Example: C:\Program Files\Microsoft Visual Studio\Common\Msdev98\Bin

< Directory where Microsoft program files are located >\Common\Tools

Example: C:\Program Files\Microsoft Visual Studio\Common\Tools

< Directory where Microsoft program files are located >\Vc98\Bin

Example: C:\Program Files\Microsoft Visual Studio\Vc98\Bin

< Directory where Microsoft DevStudio is located>\SharedIDE\Bin\Ide

Example: C:\Program Files\DevStudio\SharedIDE\Bin\Ide

< Directory where Microsoft DevStudio is located>\SharedIDE\Bin

Example: C:\Program Files\DevStudio\SharedIDE\Bin

< Directory where JD Edwards EnterpriseOne is located>\System\Bin32

Example: D:\B9\System\Bin32

In an Microsoft Windows NT environment, binaries are not compatible between the client and server machine. Do not copy .dll files or .exe files compiled on an NT workstation to an NT server. The struct alignments required by the JD Edwards EnterpriseOne server and the JD Edwards EnterpriseOne client are different.

Setting Up an Environment for GenCOM on Microsoft Visual Studio.NET

Setting up a Microsoft Windows NT client environment involves several steps. You should make sure that these items are set up appropriately:

- Include directories
- Lib directories
- Paths
- Basemake directory
- Bkoffice directory
- DXSDKROOT directory
- INETSDK directory

Important! Set up this environment for GenCom if you are using JD Edwards EnterpriseOne Tools 8.95 or later Tools releases. To avoid path-related issues, uninstall Visual Studio 6.0. If you require both Visual Studio.NET and Visual Studio 6.0, avoid path-related issues by ensuring that the Visual Studio.NET items precede the Visual Studio 6.0 items.

Example: Include Directories

<Directory where Microsoft Visual Studio .NET files are located>\include

Example: C:\Program Files\Microsoft Visual Studio .NET 2003\Vc7\PlatformSDK\Include

Example: C:\Program Files\Microsoft Visual Studio .NET 2003\Vc7\atlmfc\include

Example: C:\Program Files\Microsoft Visual Studio .NET 2003\Vc7\include

<Directory where JD Edwards EnterpriseOne is located and release either Master, Prod, or Pristine>\include

Example: C:\B9\System\include

Example: C:\B9\System\includev

Example: C:\B9\STAGINGA\include

Example: C:\E11\System\include

Example: C:\E11\System\includev

Example: C:\E11\STAGINGA\include

Example: Lib Directories

< Directory where Microsoft Visual Studio .NET files are located>\lib

Example: C:\Program Files\Microsoft Visual Studio .NET 2003\SDK\v1.1\Lib\

Example: C:\Program Files\Microsoft Visual Studio .NET 2003\Vc7\lib

Example: C:\Program Files\Microsoft Visual Studio .NET 2003\Vc7\atlmfc\lib

< Directory where JD Edwards EnterpriseOne is located>\System\Lib32

Example: C:\B9\system\Lib32

Example: C:\E11\system\Lib32

Example: Paths

< Directory where Microsoft Visual Studio .NET files are located>

Example: C:\Program Files\Microsoft Visual Studio .NET 2003\SDK\v1.1\Bin

Example: C:\Program Files\Microsoft Visual Studio .NET 2003\Common7\Tools

Example: C:\Program Files\Microsoft Visual Studio .NET 2003\Common7\Tools\Bin

Example: C:\Program Files\Microsoft Visual Studio .NET 2003\Vc7\bin

Example: C:\Program Files\Microsoft Visual Studio .NET 2003\Vc7\PlatformSDK

< Directory where Windows NT is located>

Example: C:\Winnt\System32

Example: Basemake Directories

Example: C:\Program Files\Microsoft Visual Studio .NET 2003\Vc7\PlatformSDK\Include\BKOffice.Mak

Example: Bkoffice Directories

Example: C:\Program Files\Microsoft Visual Studio .NET 2003\Vc7\PlatformSDK

Example: DXSDKROOT Directories

Example: C:\Program Files\Microsoft Visual Studio .NET 2003\Vc7\PlatformSDK

Example: INETSDK Directories

Example: C:\Program Files\Microsoft Visual Studio .NET 2003\Vc7\PlatformSDK

Running GenCOM

You run GenCOM from the command line to expose objects through COM. In a development environment, developers may run the COM Generation tool. In a production environment, a system administrator should run the COM Generation Tool.

When you use GenCOM, use the iJDEScript scripting language to script code generation activities. The syntax is:

```
GenCOM [options] [libraries]
```

For example, if you want to see available libraries that you can run GenCOM against, you enter the command`C:\B9\System\Bin32>gencom /ListLibraries` from the system command line.

To generate COM wrappers for Category 1 business functions in the CAEC library, enter this command from the command line:

```
GenCOM /Cat 1 /UserID Devuser1 /Password Devuser1 /Environment ADEVHP02 CAEC
```

Options available for generation include:

Option	Description
/?	Lists the options available for generation.
/C++ <option>	Provides GenCOM with the compiler options you want to use in the generation of the COM servers.

Option	Description
/Cat <category>	Tells GenCOM to generate wrappers based on these categories: master business functions major business functions minor business functions uncategorized business functions
/CL <file>	Tells GenCOM what compiler (.exe) to use for compilation.
/Cmd *	Processes code generation commands from the console.
/Cmd <filename>	Processes code generation commands from <filename>.
/Debug	Builds debug information (.pdb and .bsc files) into the libraries so that the Visual Studio debugger can access source information.
/EnvironmentID <env>	Provides GenCOM with the environment in which you want to sign in to JD Edwards EnterpriseOne.
/ErrFile <file>	Provides GenCOM with the filename to log errors produced by GenCOM during the generation process, for example, errors.log.
/MIDL	Provides GenCOM with the MIDL compiler options you wish to use in the generation of the COM servers.
/MTL <file>	Tells which MIDL compiler (.exe) to use for compilation.
/ListLibraries	Lists all the available libraries against which you can run GenCOM.
/MsgFile <file>	Provides GenCOM with the filename to log messages produced by GenCOM during the generation process, for example, messages.log.
/NoBSFN	Tells GenCOM not to create wrappers for business functions. This option is for generating parameter sets only.
/NoCompile	Tells GenCOM to generate the source files without compiling.
/NoDebug	Optimizes libraries for space using the /O1 Visual C++ compiler option.
/Out <path>	Provides GenCOM with the directory path in which to place the output files, for example C:\winnt\system32.
/OWRelease flag for GenCOM	You can override the OWRelease information by activating this flag and typing a string that specifies the version information. The recommendation is that you follow a naming convention that is consistent throughout the implementation or use the default version information that is generated by GenCOM.
/Password <password>	Provides GenCOM with the password with which you want to sign in to JD Edwards EnterpriseOne.
/Role	Provides GenCOM with the role with which you want to sign in to JD Edwards EnterpriseOne.

Option	Description
/STA	Generates STA components. (By default, all generated components are MTA and are optimized for scalability and performance. /STA enables you to generate STA components if you need them.)
/TempOut <path>	Provides GenCOM with the directory path in which to place temporary files needed for the build process, for example, C:\temp.
/UserID <userid>	Provides GenCOM with the user name with which you wish to sign in to JD Edwards EnterpriseOne.

Using GenCOM Output

The output for GenCOM produces fully functional COM servers based on the library to which you generate wrappers. Because you are interacting with the JD Edwards EnterpriseOne system, you must follow security and installation procedures to gain access to the system.

You must have a fully licensed copy of JD Edwards EnterpriseOne properly installed on the target machine. You must also sign in to the JD Edwards EnterpriseOne environment. For the sign-in process, you use the jdeCOMConnector interface.

Visual Basic

This code example demonstrates how to use a generated COM business function wrapper in Visual Basic. This example creates business objects. Refer to the AddressBook sample included with the COM interoperability software for a complete working example of this functionality.

```
Dim WithEvents OW As OneWorldInterface '//OneWorldInterface
Dim conn As New Connector '//COM Connector
Dim connRole As IConnector2 '//Connector Interface with role
Dim AB as JDEAddressBook '//AddressBook
Dim phone as D0100032 '//Data Source
Dim Mailing As D0100031 '//Data Source
Dim AddressAs D0100033 '//Data Source
Dim EffectiveDate As D0100019 '//Data Source
DimParentAddress As D0100381 '//Data Source
Dim sessionID As Long '//server Session ID
Private Sub Form_Load()
Set connRole = conn
'sessionID=conn.Login("Foo", "Bar", "DV9NIS2", "*ALL")
sessionID=connRole.Login("Foo", "Bar", "DV9NIS2", "*ALL")
Set OW = conn.CreateBusinessObject("OneWorld.FunctionHelper.1", sessionID)
Set AB = conn.CreateBusinessObject("AddressBook.JDEAddressBook", sessionID)
Set phone = AB.CreateGetPhoneParameterset
Set Mailing = AB.CreateGetMailingNameParameterset
SetAddress = AB.CreateGetEffectiveAddressParameterset
Set EffectiveDate = AB.CreateGetABEffectiveDateParameterset
Set ParentAddress = AB.CreateGetParentAddressParameterset
End Sub
```


Visual C++

This Visual C++ code example demonstrates how to create the connector and how to create a business function on the COM server. This example creates an AddressBook business function and uses GenCOM objects from C++.

```
#include <windows.h>
#include <stdio.h>
#include <objbase.h>
#include <comdef.h>
#include <wchar.h>
#include addressbook.h
#include AddressBook_i.c
#include jdecomconnector2.h
#include jdecomconnector2_i.c
#define IPhone ID0100032
#define IMailing ID0100031
#define IAddress ID0100033
#define IEffectiveDate ID0100019
#define IParentAddress ID0100381
#define SERVER OLESTR("COMSRV") //Change to the COM server.
#define ABNO 4242 //change this according to user input.
HRESULT CreateConnector( IConnector **ppConnector )
{
  HRESULT hr = E_FAIL;

  *ppConnector = 0;

//NOTE: Pass a COSERVERINFO struct to activate on a remote machine
  COSERVERINFO csi = {0, SERVER, 0, 0};
  MULTI_QI mqi = { &IID_IConnector, 0, 0 };
  hr = CoCreateInstanceEx(CLSID_Connector, 0, CLSCTX_LOCAL_SERVER,
         0, // &csi,
         1, &mqi);

  if(SUCCEEDED(hr) && SUCCEEDED(mqi.hr))
  {
    ppConnector = reinterpret_cast<IConnector*>(mqi.pItf);
  }
return hr;
}

HRESULT Login( IConnector **pConnector, IOneWorldInterface **ow,
long *accessno )
{
  HRESULT hr;
  IDispatch *idsptch = 0;

  printf("Login started\n");
  bstr_t User(L "Foo "), PassWord(L"Bar "), Env("DV9NIS2");
hr = (*pConnector)->Login(User,PassWord,Env,accessno );
```

```
if( !SUCCEEDED(hr))
{
  printf( "Login failed with hr = %x",hr);
  return E_FAIL;
}
_bstr_t bo("OneWorld_FunctionHelper.1");
hr=(*pConnector)->CreateBusinessObject(bo, *accessno, &idsptch );
if( !SUCCEEDED(hr)||(!ow))
{
 Printf("CreateBusinessObject(OneWorld.FunctionHelper.1) failed
with hr %x",hr);
 return E_FAIL;
}
hr=idsptch->QueryInterface(IID_IOneWorldInterface, (void **)ow );
if(!SUCCEEDED(hr)||(!ow))
{
 Printf( QueryInterface for IOneWorldInterface failed with hr "%x",hr);
 return E_FAIL
}
printf("Login completed \n");
return S_OK;
}
HRESULT UseAddressBook(IConnector *pConnector, IOneWorldInterface
*ow, long*accessno)
{
HRESULT hr;
IJDEAddressBook *ab;
IDispatch *idsptch;
IPhone *phone;
IMailing *Mailing;
IAddress *Address;
IEffectiveDate *EffectiveDate;
IParentAddress ParentAddress;

printf("Starting to use AddressBook\n");
_bstr_t bo("AddressBook.JDEAddressBook");
hr = pConnector->CreateBusinessObject(bo, *accessno, &idsptch);
hr = idsptch->QueryInterface( IID_IJDEAddressBook, (void **&ab);

if(!SUCCEEDED(hr)||(tab))
{
 printf( "CreateBusinessObject( AddressBook ) has failed with hr %x",
hr);
 return E_FAIL;
}
return S_OK;
}
```

This code creates the connector object and uses it to create a business function with its associated ParameterSet. The code then calls a method, Foo1, on the business object with the ParameterSet, the connector, and the access code returned by the act of logging on to the connector.

```
Int main(int argc, char *argv[])
{
HRESULT hr;
IOneWorldInterface *ow;
long accessno;
IConnector *pConnector;
hr - CoInitializeEx(0, COINIT_MULTITHREADED);
if(SUCCEEDED(hr))
(
 hr = CreateConnector(&pConnector);
 if(SUCCEEDED(hr))
 {
 Login( &pConnector, &ow, &accessno );
//Do more processing with AddressBook and logoff at the end.
}
 CoUninitialize();
}
```

COM Wrapper CheckVer

You can run CheckVer to verify whether a previously generated COM object is compatible with another environment. Typically, a system administrator performs this task.

The XML files generated by GenCOM are the signatures of the objects generated against specific JD Edwards EnterpriseOne environments. These XML files can be used with CheckVer to verify that the wrappers on the COM server are compatible with these environments.

When you introduce a new JD Edwards EnterpriseOne environment, you run GenCOM against the new environment by using the /NoCompile option. You also use the iJDEScript that you used to generate the wrappers on the COM server to generate XML signature files for the objects in the new environment. Run CheckVer on the COM server with the newly generated XML files to verify that the new environment is compatible with wrappers on the COM server that was previously generated with a different environment. CheckVer updates the registry settings for the wrapper on the COM server according to the result of the compatibility test. If the new environment is incompatible, the COM client cannot create business objects with the new environment.

Running CheckVer

CheckVer compares the XML signature file that is produced from GenCOM with the spec definitions on the local JD Edwards EnterpriseOne client machine. You can run CheckVer from the command line on the COM server, or CheckVer can be run automatically as part of the GenCOM process.

To see the options that CheckVer provides, run this command from the command line:

c:\>CheckVer.exe -?

Syntax

CheckVer [option] <filename>

Example

```
CheckVer -r addressbook.xml
```

Options

-r -- CheckVer reports only whether the environment is compatible with the server. It does not update the registry settings for the wrapper on the COM server with the result, and CheckVer does not validate the wrapper DLL.

CHAPTER 4

Deploying the COM Solution for Business Function Execution

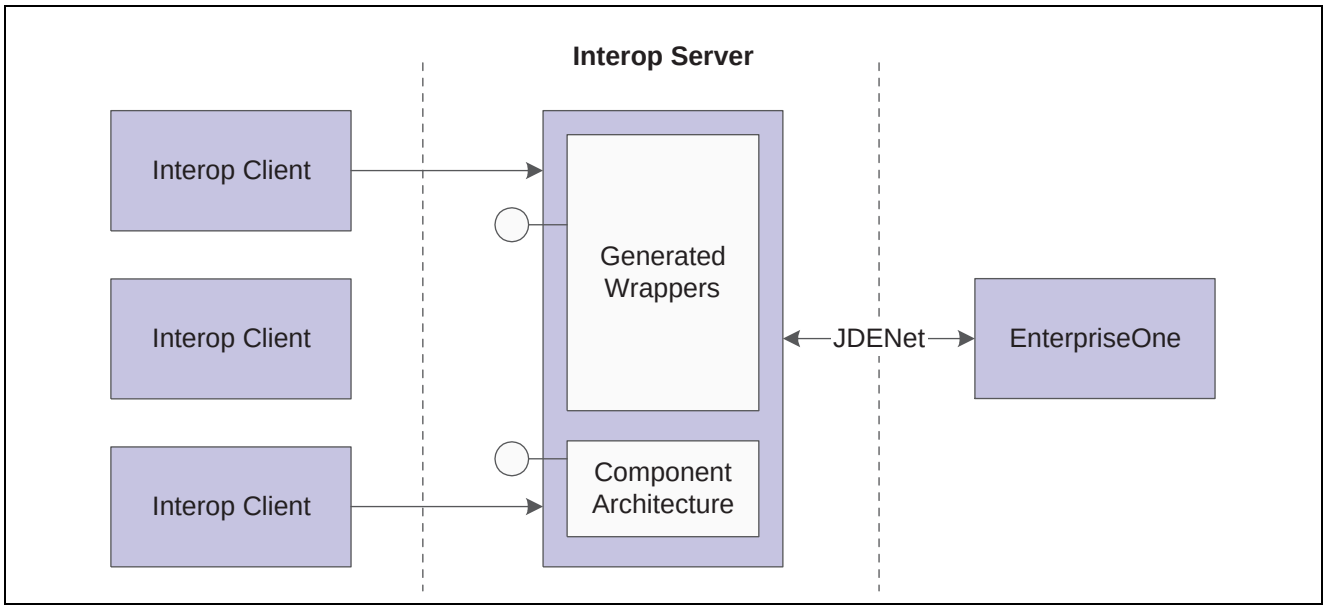
This chapter provides an overview of COM server deployment for business function execution and discusses how to:

- Set up DCOM server
- Install COM connector
- Use OCM with COM Connector
- Use BHVRCOM with COM
- Use IJDETimeZone interface
- Request Inbound XML
- Use COM reliability
- Use COM tracing and logging

Understanding COM Server Deployment for Business Function Execution

The COM server uses socket-based middleware to access the JD Edwards EnterpriseOne application server. The jdeinterop.ini file must be configured to specify the JD Edwards EnterpriseOne server. The COM server reads the jdeinterop.ini file and opens the socket connection to the specified application server.

This diagram illustrates COM server deployment:



COM server deployment

Setting Up the DCOM Server for Business Function Execution

This section provides an overview of the DCOM server and discusses how to:

- Set up DCOM for a server environment.
- Set up security on the COM server.
- Set up the identity as interactive user.
- Set up DCOM for a client environment.

Understanding DCOM Server Set Up

You can set up a DCOM server on a JD Edwards EnterpriseOne server machine. DCOM enables COM objects in a distributed environment. To ensure that the interoperability client works properly, you must set up DCOM for both a server environment and for a client environment.

Setting Up DCOM for a Server Environment

Use these steps to set up DCOM for a server environment:

1. Run GenCOM on a JD Edwards EnterpriseOne client machine, with these options:

   gencom /out <path> /tempout <path> /cmd App.cmd

   Because GenCOM is a JD Edwards EnterpriseOne client-side only tool, you must perform this step on a JD Edwards EnterpriseOne client machine.
2. Copy the App.dll file and the App.tlb file generated by GenCOM to the COM server machine.
3. On the COM server machine, from the command line:
   - Run jdecomconnector2.exe /RegServer.

- Run regsvr32 App.dll.
- Set the correct security level for jdecomconnector2.exe and App.dll.

Setting Up Security on the COM Server

Use these steps to set up security on the COM server:

1. From the Start menu, select Run.
2. Enter Dcomcnfg.exe.
3. On Distributed COM Configuration Properties, click the Default Security tab.
4. Click the Edit Default Button in Default Access Permissions group.

   The Registry Value Permissions form appears. Some entries might already be present.
5. On Registry Value Permissions, click Add.
6. On Add Users and Groups, select the appropriate domain from the List Names From option.
7. Click Everyone, and then click Add.

   Type of access should be Allow Access.
8. Click OK.

   Repeat Steps 4 through 7 for default launch permissions. No setup is required for default configuration permissions.

Setting Up the Identity as Interactive User

Use these steps to set up the identity as interactive user:

1. Run DCOMCnfg.
2. On Distributed COM Configuration Properties, select JDECOMConnector2, and then click Properties.
3. On JDECOMConnector2Properties, click the Identity tab, and then select the interactive user option.
4. Click Apply to apply the change.

Note. You must perform this task every time you register the connector. If you copy the JDECOMConnector2.exe using Explorer, Explorer reruns the registration, and you must repeat these steps.

To use Callbacks (Connection Points) with the COM solution, repeat the same procedure on the COM client machine. Most of the shipped examples use Callbacks and require that you open the security on the client machine.

Setting Up DCOM for a Client Environment

Use these steps to set up DCOM for a client environment:

1. From a DOS prompt on the DCOM client machine, run jdecomconnector2.exe /RegServer.
2. At the prompt, enter oleview.exe.
3. From the menu bar, select oleview.
4. Click View and select Expert Mode.
5. In the oleview window under Object Classes, double-click All Objects, and wait for all objects to appear.

6. Under All Objects, find and click Connector Class.
7. Click the Implementation tab on the right-side panel, and then click the local server and remove anything that appears in the editing window.
8. On the Activation tab, select the Launch as Interactive User option.
9. In Remote Machine Name, enter the COM server machine name.
10. Repeat steps 5 through 8 for MathNumeric Class.
11. Start the DCOM client application.

Note. Client-only business functions are not reachable.

Installing COM Connector

This section discusses how to install the COM connector in a non-JD Edwards EnterpriseOne client environment.

Installing COM Connector on a Non-JD Edwards EnterpriseOne Client Environment

Use these steps to install the COM connector on a non-JD Edwards EnterpriseOne client machine:

1. Copy these files from the JD Edwards EnterpriseOne server (system\bin32) to a directory on the desired machine. For example, copy the files in c:\program files\JDEdwards to a non-JD Edwards EnterpriseOne client machine.
   - JDECOMConnector2.exe
   - JDECOMMN.dll
   - callobject.dll
   - comlog.dll
   - EventManager.dll
   - OneWorldInterfaceTx.dll
   - xmlinterop.dll
   - jdel.dll
   - jdethread.dll
   - jdeunicode.dll
   - ustdio.dll
   - icuil8n.dll
   - jdeinterop.ini to c:\(root directory)
   - checkver.exe
   - ICUUC.dll
   - Icu\data*.*
   - EventClass.dll

- EventListener.dll
- ClientService.dll - JD Edwards EnterpriseOne Tools 8.95 and later Tools releases only
- EventHandler.dll - JD Edwards EnterpriseOne Tools 8.95 and later Tools releases only
- JMSEvent.dll - JD Edwards EnterpriseOne Tools 8.94 only

2. Create a new directory Icu\data\ on the machine where the COM server is located. Copy all of the files from the JD Edwards EnterpriseOne server in folder system\Locale\xml*.* into Icu\data\. Create a new system variable, *ICU_DATA,* in the environment variables of the system properties and specify the path to the Icu\data\ as the value.
3. Execute this command on the target location to register the COM connector components:

   ```
   JDECOMConnector2.exe /RegServer
   ```

4. Run GenCOM on a JD Edwards EnterpriseOne client machine and copy the output DLL and the wrapper components (for example, wrapper.dll).
5. Execute this command to register the COM wrapper components:

   ```
   regsvr32wrapper.dll
   ```

6. Create the JDEinterop.ini file.

   Set the JD Edwards EnterpriseOne server and port values to the JD Edwards EnterpriseOne application server with which you want the COM server to communicate.

   The COM server is now ready.

To unregister the COM server, use the /unreserved option. For example:

```
JDECOMConnector2.exe /unreserved
```

To unregister the COM wrapper, use the /u option. For example:

```
regsvr32 /u wrapper.dll
```

See Also

Chapter 7, “Understanding jdeinterop.ini for COM Connector,” page 75

Using OCM Support with COM Connector

You use Object Configuration Manager (OCM) to map business functions to a JD Edwards EnterpriseOne server so that the COM connector can access OCM to run business functions. You no longer configure the jdeinterop.ini file to define the JD Edwards EnterpriseOne server from which you want to execute business functions. Using OCM support should result in increased performance, scalability, and load balancing. OCM mapping enables the COM interoperability server to distribute the processes of the COM connector client to various JD Edwards EnterpriseOne servers’ requests, depending on the user, environment, and role name.

To take advantage of COM connector OCM support, the system administrator should:

- Get the GenCOM JD Edwards EnterpriseOne 8.9 (or later) version and regenerate the business wrapper function.
- Configure the OCM and map the business function on the enterprise server.
- Add these settings in the jdeinterop.ini configuration file.

[INTEROP]

Setting	Explanation
EnterpriseServer = ntropt1	For COM events and backward compatibility.
SecurityServer = ntropt1	Validates the login.
Port = 6079	The port number.

The database administrator or JD Edwards EnterpriseOne administrator can provide these settings for the [OCM] section of the jdeinterop.ini configuration file. This information is used for database connectivity.

[OCM]

Setting	Explanation
DSN=ODA ITTND17	The data source name from the system DSN of the ODBC setting.
OCM Datasource = COM OCM	System data source for JD Edwards EnterpriseOne client.
DB User = JDE	User for the data source connection.
DB Pwd = JDE	Password for the data source connection.
Object Owner = SYS9	For UNIX platforms, this is the object owner in the [DB SYSTEM SETTINGS].
Seperator=.	For Oracle, SQL and UDB databases, the separator is a period (.); for iSeries, the separator is a slash (/).

If you use a client machine, the settings can be found in the client jde.ini file. An example of the database name and object owner is: JDE9.SYS9, where JDE9 is the database name and SYS9 is the object owner.

See Also

JD Edwards EnterpriseOne Tools 8.96 Configurable Network Computing Implementation Guide, "Working with Object Configuration Manager," Understanding Object Configuration Manager

Using BHVRCOM with COM

JD Edwards EnterpriseOne clients use the BHVRCOM structure to control the execution of business functions. A COM client can use the IBHVRCOM interface to set and get BHVRCOM values for business functions. The interface definition is in the jdeconnector2.idl file.

This Visual Basic code demonstrates how to query the IBHVRCOM interface and pass values to business functions:

```
Dim conn As New Connector  '//COM Connector
DIM WithEvents OW As OneWorldInterface  '//OneWorldInterface
Dim myBHVRCOM As IOneWorldBHVRCOM  '//BHVRCOM
```



```
Dim AB As JDEAddressBook  '// AddressBook
Dim phone As D0100032  '//Data source
1 = conn.Login("JDE", "JDE", "M7332RS02")
Set OW = conn.CreateBusinessObject("OneWorld.FunctionHelper.1",1)
Set myBHVRCOM = OW  '// query the IOneWorldBHVRCOM interface
MyBHVRCOM.iBobMode = 8  '// set BHVRCOM values
MyBHVRCOM.szApplication = "myApp"
MyBHVRCOM.szVersion = "myVersion"
Set AB = conn.CreateBusinessObject("AddressBook.JDEAddressBook",1)
Set phone = AB.CreateGetPhoneParameterset
Phone.mnAddressNumber = 1
AB.GetPhone phone, OW, conn, 1  '// business function is executed with
the BHVRCOM values
```

This table explains some of the code:

Code	Explanation
myBHVRCOM.iBobMode=	BobMode is the mode (add, update, delete) of the interactive application. Values for BobMode are: BOB_MODE_UNDEFINED = 0 BOB_MODE_SPECIAL = 1 BOB_MODE_ADD = 2 BOB_MODE_ADD_PRIMARY = 3 BOB_MODE_ADD_SPECIAL = 4 BOB_MODE_DELETE = 5 BOB_MODE_UPDATE = 6 BOB_MODE_UPDATE_SPECIAL = 7 BOB_MODE_INQUIRE = 8 BOB_MODE_COPY = 9
myBHVRCOM.szApplication=	The value is the name of the interactive application.
MyBHVRCOM.szVersion=	The value is the version of the interactive application. This field can be used for localizations of the applications.

Use IJDETimeZone Interface

To modify and display the JDEUTIME data type in the appropriate format, the COM client and GenCOM must use the JDEUTIME APIs. Date and time information is displayed in a time based on the date and time that is in the personal profile or a time zone specified by an application.

These steps, along with sample code, illustrate how to use the IJDETimeZone Interface.

- Create the IJDETimeZone interface.

```
MULTI_QI mqi = { &IID_IJDETimeZone, 0, 0 };
hr = CoCreateInstanceEx(CLSID_JDETimeZone, 0, CLSCTX_ALL, 0, 1, &mqi);
if (SUCCEEDED(hr) && SUCCEEDED(mqi.hr))
{
   *ppJdeTimeZone = reinterpret_cast<IJDETimeZone*>(mqi.pItf);
}.
```

- Set the time for a time zone (UTC-5:30) for the data structure DXXXXXX.

  If a time zone is not specified, the time is considered to be at UTC. If an invalid time zone string is passed, then an error occurs.

```
DATE dt;
BSTR bstrUTC = SysAllocString(L"UTC-5:30");
pJDETimeZone->put_DateTime(bstrUTC,&dt);
DXXXXXX->put_jdOrderDate(pJDETimeZone);
```

- Get a time for a given time zone from JD Edwards EnterpriseOne.

  If a time zone string is not passed, the time and date stored in JD Edwards EnterpriseOne, which is at UTC, is returned. If an invalid time string is passed, then an error occurs.

```
DXXXXXX->get_jdOrderDate(pJDETimeZone);
DATE dt;
BSTR bstrUTC = SysAllocString(L"UTC-5:30");
pJDETimeZone->get_DateTime(bstrUTC,*dt);
```

XML File generated by GenCOM for IJDETimeZone

For each data item whose data type is JDEUTIME in the data structure DXXXXXX, GenCOM generates this XML file:

```
<Signature environment="Environment Name">
 <Interface name="Interface Name">
    <Method name="BSFN">
        <Param name="DXXXXXX" type="u" />
    </Method>
 </Interface>
</Signature>
```

Requesting Inbound XML Using COM Server

You can use the COM connector to send inbound synchronous XML requests (such as XML CallObject, XML List, and XML Transaction) to the JD Edwards EnterpriseOne server. The COM connector uses XML APIs to access the inbound XML requests.

Using COM Reliability

Graceful fail-over and fault tolerance mechanisms are important, especially for applications that require high availability. The COM connector provides basic support for fault tolerance at the protocol level.

You should take additional precautions to provide further reliability. After you use the COM connector to enter an order or execute a business function, the process should:

- Handle transaction failures.

  Transactions can fail because of communication line failures. Sometimes transactions must be aborted because of errors in input or deadlocks. These failures must be handled appropriately.

- Wait for the confirmation or success notification from the business function to ascertain that the call was successfully committed.

- Query on the order entered to make sure that it has been committed to the database.

  Due to high network traffic, a business function can properly execute, but the confirmation message might not reach you.

Using COM Tracing and Logging

You use COM tracing and logging to help you debug the COM applications. You use the jdeinterop.ini file to configure tracing and logging settings. The logging format is similar to the JD Edwards EnterpriseOne logging format. For example, both logging formats include the Time Thread ID [User ID] and Description, as illustrated:

Thu Mar 02 14:48:01 2000 294 [AR618238] Failed to Login to Environment <ADEVHPO2>

Errors are written to the JobFile and trace messages are written to the Debug File. When trace is enabled, error messages go into both trace and error logs.

You can change the jdeinterop.ini settings while the connector is running by completing these the steps:

1. Modify the jdeinterop.ini file.
2. Right-click the Connector System Tray button.
3. Select the menu item ChangeIniSettings.

   If an option in the jdeinterop.ini file does not have an entry, the default value is used.

Resolving Tracing Issues

Tracing affects performance. You do not need to use tracing unless you are debugging an application. If performance is negatively affected, ensure that the tracing level is set to zero.

If no logs are generated, complete these steps:

- Ensure that you have specified the proper path in the ini file.
- Verify that disk space and the permissions on the file system are correct.
- Verify whether the default log files have been generated.
- Check the interop.log to see if any errors corresponding to logging have been generated.

- Check the interop.log file to see if the ini settings that are being used are the same as what you have specified elsewhere.

CHAPTER 5

Using COM Transactions

This chapter provides an overview of COM Interoperability Transactions and discusses how to:

- Set up the COM+ Environment.
- Run a COM+ Transactions.
- Run a Distributed Transaction.

Understanding COM Interoperability Transactions

COM interoperability transactions include COM connector prepare, commit, and rollback functionality. The COM transaction interoperability solution supports these types of transactions:

- Auto commit transactions
- Manual commit transactions

A transaction can be started as auto commit or manual commit. In auto commit, JD Edwards EnterpriseOne automatically commits the transaction that has been started. If a transaction is started in manual commit, you have to explicitly call prepare and commit functionality for the transaction to be committed.

The COM connector also supports manual commit. Typically, a transaction is started in manual commit by calling BeginTransaction with the flag set to 1. Subsequent calls to prepare and commit commits the transaction. The COM connector prepare and commit does not support distributed transactions that involve transactions other than JD Edwards EnterpriseOne.

Outline for Calling Prepare and Commit

This table provides an outline for calling prepare and commit:

Function	Description
Dim soeOWInterface As OneWorldInterface	Declare the OneWorldInterface.
soeOWInterface.BeginTransaction (accessNumber, connector, txMode)	Start the transaction in manual commit by calling begin transaction and setting the txMode to 1. 0 is for auto commit.
//execute all BSFNs like the //enddoc and other BSFNs	After a call to Begin Transaction is made, do all the transactions that you want to enclose within this manual commit before calling prepare.

Function	Description
soeOWInterface.Prepare	Call prepare when all of the transactions are done.
soeOWInterface.Commit (or) soeOWInterface.RollBack	Call Commit to commit the transaction (or) Rollback to roll back the transaction if an error occurs.

The default timeout value for a manual transaction is 5 minutes. If you do not commit the transaction within 5 minutes, the transaction context is freed and the transaction is rolled back. You can change the default timeout by setting the manual_timeout value in the [INTEROP] section of the jdeinterop.ini file. The value is in milliseconds.

COM+ Two-Phase Commit Transaction

The COM connector can participate in distributed transactions. The COM connector's ability to participate in distributed transactions enables any application that uses the COM connector to participate in the two-phase commit transaction. Applications that have the capability to participate in distributed transactions can also use the COM connector.

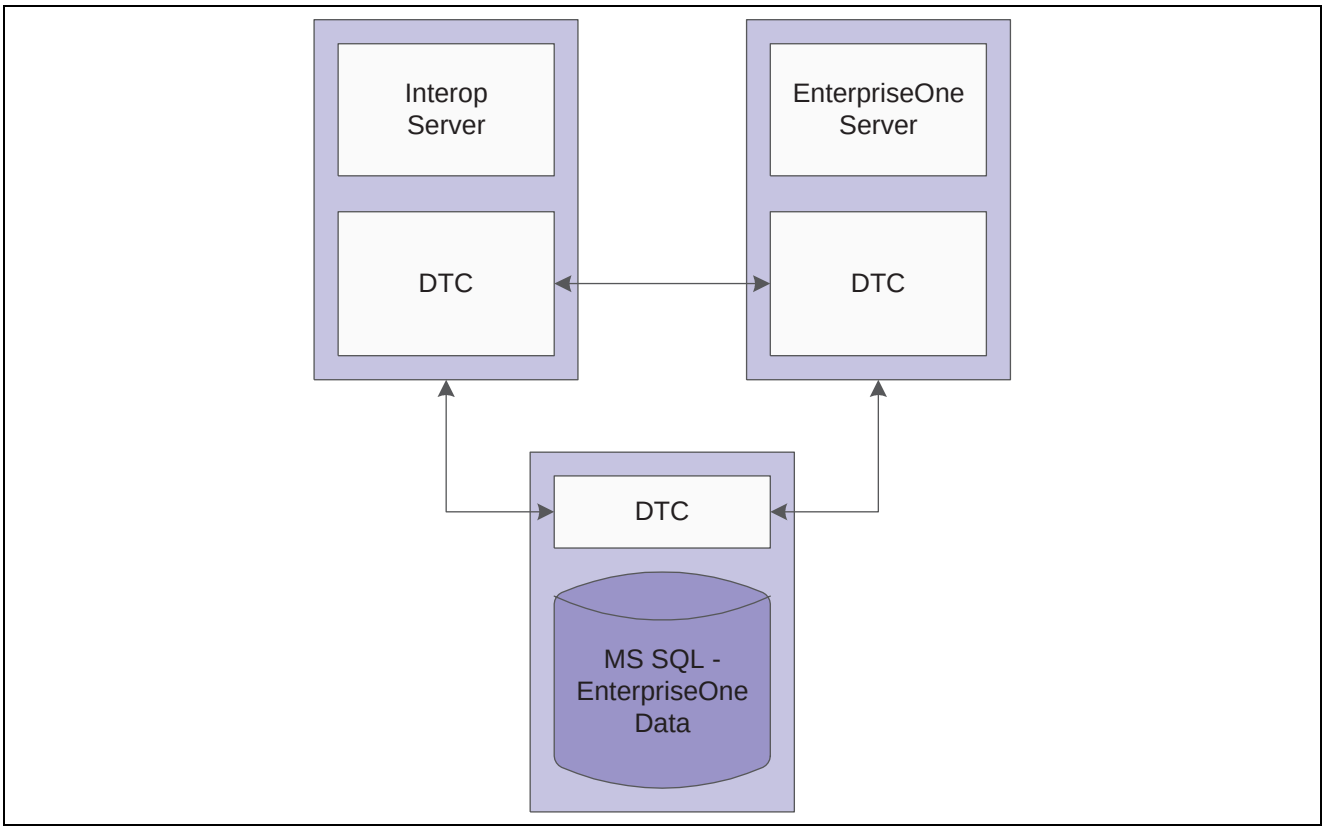
Setting Up the COM+ Environment

Typically, when you use COM+ for two-phase commit, you must set up the environment for these three computers:

- COM connector
- JD Edwards EnterpriseOne server
- Database server

A distributed transaction coordinator (DTC) is expected to run on each of the machines. Before testing the COM+ two-phase commit, you must make sure that the DTCs on each machine are correctly configured and that the DTCs talk to each other.

This illustration shows the physical configuration:



COM+ Environment Configuration

Note. Typically, administrative rights are required for you to run the examples, which talk to DTCs on different machines. For more information about setting up DTC and various configurations, refer to the Microsoft documentation.

Running a COM+ Transactions

This section provides an overview of JD Edwards EnterpriseOne participating in a COM+ transaction and discusses how to:

- Create a Transactional Object
- Create a Transactional Client

Understanding COM+ Transactions

This code outline explains how to develop code for COM connector and JD Edwards EnterpriseOne participation in COM+ transactions:

Code	Explanation
Dim ow As OneWorldTx	Declare new OneWorldTx.
Set ow = New OneWorldTx ow.Initialize laccessNumber, connRole	Initialize the transaction by passing the access number returned from a successful logon and the connector.

Code	Explanation
ow.BeginTransaction laccessNumber, connRole, 1	Start a transaction in Manual Commit. 1 Manual commit 0 Auto Commit
EditLine, EndDoc	Do all the processing here like `BeginDoc`.
GetObjectContext().SetComplete or GetObjectContext().SetAbort	Use `SetComplete` to commit the transaction through DTC or use `SetAbort` to abort the transaction.

Note. In COM+, an AutoCommit attribute exists that implicitly commits a transaction if no errors exist. This attribute is in the Component Services Administration tool. However, if an explicit call to SetAbort is made, the transaction aborts.

These code examples illustrate how to create a sales order entry transactional object (SOETxObject) and a sales order entry transactional client (SOETxClient). After you create the transactional object and transactional client, you can run the transactions. Use these steps to run a sales order entry transaction in COM+ where the COM connector and JD Edwards EnterpriseOne participate:

1. Run the SOETxObject.
2. Run the SOETxClient.
3. Note the Sales Order Entry number that is displayed.
4. When the message box appears for Commit or Abort, select the appropriate action.
5. Verify in JD Edwards EnterpriseOne whether the sales order has been entered. The sales order should be entered only when committed.

Creating a Transactional Object (SOEProj.vbp)

This sample code shows how to create a SalesOrderEntry transactional object (SOETxObject => SOEClass2.cls).

```
Public Sub run()
On Error GoTo errorhandler
Dim ow As OneWorldTx

 Dim bhvr As IOneWorldBHVRCOM

 Dim conn As New Connector           '// COM Connector
 Dim connRole As IConnector2         '// Connector Interface with Roles

 Dim soeObject As JDESalesOrderEntry '// SalesOrderEntry
 Dim soeBeginDoc As D4200310H
 Dim soeEndDoc As D4200310G
 Dim soeEditLine As D4200310F
 Dim soeClearWF As D4200310I
```



```
Dim s As String
Dim d As New MathNumeric
Dim mnQuanityOrdered As New MathNumeric
Dim mnUnitPrice As New MathNumeric
Dim response

Dim laccessNunber As Long

' Name Information
Dim strComputerName As String
Dim lngNameLength As Long

Const WRITE_FLAG = "2"

Dim i As Boolean
Set connRole = conn
laccessNumber = connRole.Login("UserID", "PWD", "ENV", "ROLE")

Set ow = New OneWorldTx

ow.Initialize laccessNumber, connRole
'oneworld transaction initialized to manual
ow.BeginTransaction laccessNumber, connRole, 1

Set bhvr = ow
bhvr.szApplication = "COM+"
Set soeObject = connRole.CreateBusinessObject("SalesOrderEntry.
JDESalesOrderEntry", laccessNumber)
' please change the progid to correct progId
Set soeBeginDoc = soeObject.CreateF4211FSBeginDocParameterset
Set soeEditLine = soeObject.CreateF4211FSEditLineParameterset
Set soeEndDoc = soeObject.CreateF4211FSEndDocParameterset
Set soeClearWF = soeObject.CreateF4211ClearWorkFileParameterset

' Get computer name for use later
strComputerName = Space(30)
lngNameLength = 30
p_ret = GetComputerName(strComputerName, lngNameLength)
If p_ret <> 1 Then
  MsgBox (GetComputerName failed!)
  'End
Else
  strComputerName = Mid(strComputerName, 1, lngNameLength)
End If
' MsgBox (Create Biz Object Done!)

'//////////////BEGIN DOC//////////////
soeBeginDoc.Reset
soeBeginDoc.cCMDocAction = "A"
soeBeginDoc.cCMProcessEdits = "1"
```

```
soeBeginDoc.cCMUpdateWriteToWF = WRITE_FLAG
soeBeginDoc.szCMProgramID = "VB"
soeBeginDoc.szCMVersion = "ZJDE0001"
soeBeginDoc.szOrderCo = "00200"
soeBeginDoc.szOrderType = "SO"
szBUnit = "M30"
soeBeginDoc.szBusinessUnit = Space(12 - Len(szBUnit)) + szBUnit
d = Val("4242")
soeBeginDoc.mnAddressNumber = d
soeBeginDoc.mnShipToNo = d
soeBeginDoc.jdOrderDate = Date
soeBeginDoc.cMode = "F"
soeBeginDoc.szUserID = "JDE"
soeBeginDoc.cRetrieveOrderNo = "1"

If strComputerName <> "" Then
    soeBeginDoc.szCMComputerID = strComputerName
End If
' MsgBox ("Before F4211FSBeginDoc")
soeObject.F4211FSBeginDoc soeBeginDoc, ow, connRole, laccessNumber

MsgBox Round(soeBeginDoc.mnOrderNo, 0)

'//////////EDIT LINE///////////

soeEditLine.mnCMJobNo = soeBeginDoc.mnCMJobNumber
orderNum = soeBeginDoc.mnOrderNo
soeEditLine.mnOrderNo = soeBeginDoc.mnOrderNo
soeEditLine.szBusinessUnit = soeBeginDoc.szBusinessUnit
soeEditLine.szCMComputerID = soeBeginDoc.szCMComputerID
soeEditLine.cCMWriteToWFFlag = WRITE_FLAG

soeEditLine.szOrderType = soeBeginDoc.szOrderType
' Load items from UI into edit line structure
soeEditLine.szItemNo = "1001"
mnQuanlityOrdered = "2"
soeEditLine.mnQtyOrdered = mnQuanityOrdered

' MsgBox ("Before F4211FSEditLine.")
' Call business function
soeObject.F4211FSEditLine soeEditLine, ow, connRole, laccessNumber
' MsgBox ("After F4211FSEditLine.")

'///////////////ENDDOC//////////////
soeEndDoc.mnCMJobNo = soeBeginDoc.mnCMJobNumber
soeEndDoc.mnSalesOrderNo = soeBeginDoc.mnOrderNo
soeEndDoc.szOrderType = soeBeginDoc.szOrderType
soeEndDoc.szCMComputerID = strComputerName
soeEndDoc.cCMUseWorkFiles = WRITE_FLAG
'Call business function
```



```
  'MsgBox ("Before F4211FSEndDoc.")
 soeObject.F4211FSEndDoc soeEndDoc, ow, connRole, laccessNumber
 'MsgBox ("After F4211FSEndDoc.")
 MsgBoxRes = MsgBox("Do you want to abort?", vbYesNo, "Transaction
Decision")
 If MsgBoxRes = vbYes Then
   GetObjectContext.SetAbort
 Else
   GetObjectContext.SetComplete
   MsgBox ("Order Saved")
 End If

 '///////CLEAR WORK FILE////////////////

 soeClearWF.cClearDetailWF = WRITE_FLAG
 soeClearWF.cClearHeaderWF = WRITE_FLAG
 soeClearWF.mnJobNo = soeBeginDoc.mnCMJobNumber
 soeClearWF.szComputerID = strComputerName
 'Call business function
 'MsgBox ("Before F4211ClearWorkFile.")
 ow.BeginTransaction laccessNumber, connRole, 0
 soeObject.F4211ClearWorkFile soeClearWF, ow, connRole, laccessNumber
 'MsgBox ("After F4211ClearWorkFile.")
 Set soeObject = Nothing
 Set soeBeginDoc = Nothing
 Set soeEditLine = Nothing
 Set soeEndDoc = Nothing
 Set ow = Nothing
 connRole.Logoff (laccessNumber)
 Set connRole = Nothing

 Exit Sub

errorhandler:
 GetObjectContext().SetAbort
 connRole.Logoff (laccessNumber)
 Set ow = Nothing
End Sub
```

Module1 : Module1.bas

Create a module file and declare the GetComputerName function.

```
Public Declare Function GetComputerName Lib "kernel32" Alias
"GetComputerNameA" (ByVal lpBuffer As String, nSize As Long) As Long
```

Creating a Transactional Client

This sample code shows how to create a SalesOrderEntry transactional client (SOETxClient => SOETxClient.vbp):


```
'////SOETxClient////
Private Sub Form_Load()
Dim c As SOEClass2         '// VB SOE transactional object
Set c = New SOEClass2
c.run
Set c = Nothing
End Sub
```

Running a Distributed Transaction

This section provides an overview of JD Edwards EnterpriseOne participating in a distributed transaction and discusses how to:

- Create MTStest for a Distributed Transaction.
- Create ClientPrj for a Distributed Transaction.
- Register a New COM+ .dll.

Understanding COM+ Transaction

This sample code, called MTStest.vbp, shows how to create a distributed transaction using COM+. This project contains these two classes:

- MTSTestClass, which queries and updates a test SQL database
- OWTxClass, which runs the Sales Order Entry

OWTxClass is almost identical to the previous SOETxObject, except that the message box for commit or abort is no longer necessary.

MTStest.dll must be registered in the COM+ Component Services, and the transaction property should be set to required; it might have been set already.

Create a sample SQL test database table SOE2PCTest. SOE2PCTest table has two columns, SONum and LastSONum. The test selects the LastSONum and then updates the table by incrementing the previous value by 1 when commit is called.

Sample code called ClientPrj.vbp will call the transactional object.

Both of the transactions are committed by the DTC when the SetComplete call is made. The DTC aborts the transaction when the SetAbort call is made or if any part of the transaction fails.

Use these steps to run a sales order entry as a distributed transaction in COM+ where the COM connector, JD Edwards EnterpriseOne, and an SQL database participate.

1. Run the MTStest.vbp.
2. Run the ClientPrj.vbp.
3. Click the Call Database_ Test_ Method button.
4. Switch back to the MTStest and note the sales order number.
5. When a message box appears to Commit or Abort, select the appropriate action.

6. Verify in JD Edwards EnterpriseOne whether the sales order has been entered. When the transaction is aborted, the sales order should not be in JD Edwards EnterpriseOne, and the test database should not increment the count.

Creating MTStest for a Distributed Transaction (MTStest.vbp)

This code sample provides detail code for creating MTStest.

Note. This sample code has message box statements to help better understand the step-by-step flow of the code. Since DTC is managing the transactions, it is necessary not to lock the tables for a long time. When you use message boxes, you stop the program flow. When regression testing, you must remove all of the message boxes. You can write to a log file instead.

MTSTestClass : MTStest.bas

You can use this sample code to create MTStest:

```
Option Explicit
 Public Function Database_Test_Method(_ByVal szConnect As String) As String

 Dim stmt As String

 On Error GoTo errhandler

 Dim ctxObject As ObjectContext
 Set ctxObject = GetObjectContext()

 Dim MsgBoxRes
 Dim cn As ADODB.Connection
 Dim rsSelect As ADODB.Recordset
 Dim rs As ADODB.Recordset

 Set cn = New ADODB.Connection
 With cn
   .ConnectionTimeout = 10
   .ConnectionString = szConnect
   .Open
 End With
''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
' SONUM and LASTSONUM are columns created in a database called '
' COMPLUS. '
' Database server is called soe2pctest.                 '
' LASTSONUM gets incremented when commit is used.            '
' Change these values according to Database created           '
''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
 Set rs = New ADODB.Recordset
 Set rsSelect = New ADODB.Recordset
 rsSelect.Open "SELECT LASTSONUM FROM soe2pctest", cn, adOpenDynamic,
_ adLockReadOnly
 Dim i As Integer
 For i = 1 To 3
```

```
 stmt = "Update SOE2PCTest set LASTSONUM=" & rsSelect(0).Value + 1& &
" where SONUM = 1"
 cn.Execute stmt, 1, -1
 rsSelect.Close

 Dim c As OWTXClass
 Set c = New OWTXClass

 c.run

 Set c = Nothing
 cn.Close

 Set rs = Nothing
 Set cn = Nothing
 MsgBoxRes = MsgBox("Do you want to Commit?", vbYesNo, "Transaction
 Decision")
 If MsgBoxRes = vbYes Then
  ctxObject.SetComplete
 Else
  ctxObject.SetAbort
 End If
 Next I

 Exit Function

errhandler:
 Err.Raise vbObjectError, "MTSTest.MTStest.Database_Test_Method", _
Err.Description
 ctxObject.SetAbort
 Exit Function

End Function
```

Module1 : Module1.bas

Create a module file and declare the GetComputerName function.

```
Public Declare Function GetComputerName Lib "kernel32" Alias
"GetComputerNameA" (ByVal lpBuffer As String, nSize As Long) As Long
```

Creating ClientPrj for a Distributed Transaction

This code sample provides detail code for creating ClientPrj.vbp.

Note. This sample code has message box statements to help better understand the step-by-step flow of the code. Since DTC is managing the transactions, it is necessary not to lock the tables for a long time. When you use message boxes, you stop the program flow. When regression testing, you must remove all of the message boxes. You can write to a log file instead.

```
Private Sub Command2_Click()
 Dim szConnect As String
 szConnect = "Driver={SQL Server};" & _
    "Server=AServerName;Uid=UserID;Pwd=Passwd;Database=DBName"
 '(NOTE: You may need to change the connection
 ' information to connect to the database.)

 Dim obj As Object
 Set obj = CreateObject("MTStest.MTSTestClass")

 MsgBox obj.Database_Test_Method(szConnect)
 Set obj = Nothing
 Unload Me
End Sub

Private Sub Form_Load()

  Command2.Caption = "Call Database_Test_Method"

 End Sub
```

Registering the COM+ .dll

A new COM+ dll (OneWorldinterfaceTx.dll) is provided to be used along with the COM connector to participate in a two-phase commit. OneWorldInterfaceTx.dll must be registered with the COM+ component services.

Use these steps to register OneWorldInterfaceTx.dll:

1. On the PC, navigate to COM+ Applications:

   Control Panel > Administrative Tools > Component Services
2. Expand these buttons and folders:

   Component Services > Computers > My Computer
3. Select COM+ Applications.
4. Right-click COM+ Applications, select New, and then select Application.

   The COM Application Install Wizard appears.
5. On Install or Create a New Application, select Create an empty application and then click Next.
6. On Create Empty Application, enter the name of the application (OneWorldInterfaceTx) that you are registering.
7. Select an Activation type, and then press Next.
8. On Set Application Identity, select Interactive User, and then click Next.
9. Click Finish to close the wizard.
10. On the PC, expand these folders:

    COM+ Applications > OneWorldInterfaceTx
11. Select Components.

12. Right-click Components, select New, and then select Component.
13. The COM Component Install Wizard appears.
14. On Import or Install a Component, select Install New Component(s), and then click Next.
15. On Select New Files to Install, browse to the application (OneWorldInterfaceTx.dll) on the client install directory or the COM interoperability server.
16. Add the application and then click Next.
17. Click Finish to close the wizard.

    The application (OneWorldInterfaceTx.dll) is registered.
18. On the PC, expand the Components folder and then right-click the application (OneWorldInterfaceTx.dll) you just registered.
19. Select Properties.
20. On OneWorldInterfaceTx Properties, click the Transactions tab.
21. For the Transaction support field, select the Required option.
22. Click OK.
23. Close the component servers.

The COM connector should be registered using the method described in the chapter titled Installing COM Connector on a Non-JD Edwards EnterpriseOne Client Environment.

The SalesOrderEntry and other wrapper dlls should be registered using the standard RegSvr32 command.

A new transactional object that is going to participate in the COM+ transactions (for example, SOEClass2.dll) must be created and registered through the COM+ component services of the administrative tools. The transactions property of this object should be set to Required. This transactional object will use the new OneWorldInterfaceTx.dll for starting a transaction, executing a business function, and so on. The code outline is explained in Case1: JD Edwards EnterpriseOne Participates in COM+ Transaction. Detail sample code for the SalesOrderEntry transaction object (SOETxObject) is provided.

After the transactional object is created, open a new VB sample SalesOrderEntry client and call the SOEClass2 object. The VB SOETxClient code is provided.

Two cases of the Sales Order Entry application are discussed. Case 1 is when JD Edwards EnterpriseOne participates in the COM+ transaction. Case 2 is when JD Edwards EnterpriseOne participates in a distributed transaction.

CHAPTER 6

Using COM Connector Solution for Events - Guaranteed Events

This chapter provides an overview of COM connector guaranteed events and discusses how to:

- Set up the COM connector for guaranteed events using JD Edwards EnterpriseOne Tools 8.94 with JD Edwards EnterpriseOne Applications 8.11.
- Set up the COM connector for guaranteed events using JD Edwards EnterpriseOne Tools 8.95 and later Tools releases with JD Edwards EnterpriseOne Applications 8.9 and later Applications releases.
- Implement JD Edwards EnterpriseOne interfaces.
- Register EventSink for persistent subscription.

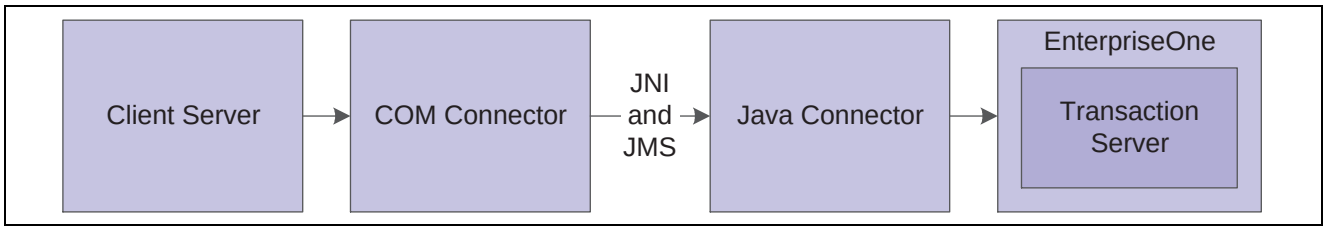
Note. This chapter is applicable only if you use guaranteed event delivery. Guaranteed event delivery is available when you use JD Edwards EnterpriseOne Tools 8.94 with JD Edwards EnterpriseOne Applications 8.11 or JD Edwards EnterpriseOne Tools 8.95 and later Tools releases with JD Edwards EnterpriseOne Applications 8.9 and later Applications releases. Headings in this chapter that have an 8.94 after them are applicable for Tools release 8.94 with Applications release 8.11 only. Headings in this chapter that have 8.95 after them are applicable for Tools release 8.95 and later Tools releases.

Refer to the Classic Events chapters if you use JD Edwards EnterpriseOne 8.93 or earlier releases, or if you use JD Edwards EnterpriseOne Tools 8.94 with JD Edwards EnterpriseOne Applications 8.10 or 8.9.

Understanding COM Connector Guaranteed Events

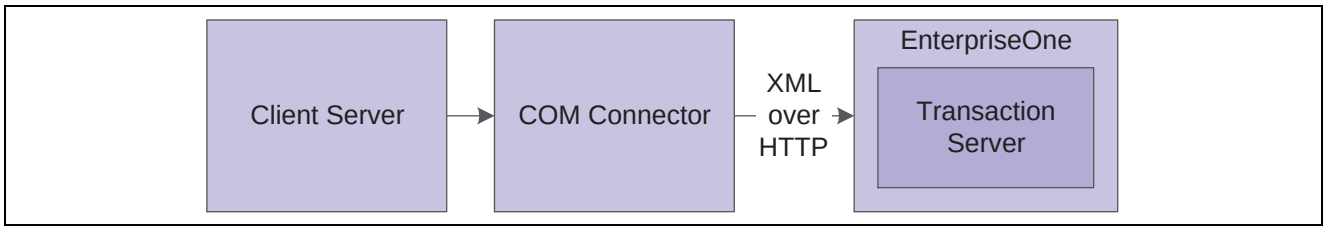
The COM connector events solution uses the Microsoft COM+ Events Service. COM+ Events Loosely Coupled Events, which matches and connects publishers and subscribers, is part of the Microsoft Windows 2000 Component Services. The EventClass is a COM+ component that contains interfaces and methods that are used by the publisher to initiate events. The EventClass manages the connection between publisher and subscribers. The EventClass.dll, which contains the IOWEvent interface, is provided. The COM servers and COM clients must implement this interface so that when an event is initiated, this interface is called by the COM+ Events Service and the implementation is executed. The implementation decides what the delivered event and the event data should do. This implementation is COM server or COM client specific.

To support guaranteed event delivery for JD Edwards EnterpriseOne Tools release 8.94, the COM connector uses the Java Connector to access JD Edwards EnterpriseOne. This illustration shows the COM connector architecture for guaranteed events using JD Edwards EnterpriseOne Tools 8.94:



COM connector architecture-guaranteed event delivery for JD Edwards EnterpriseOne Tools release 8.94

To support guaranteed event delivery for JD Edwards EnterpriseOne Tools release 8.95 and later Tools releases, the COM connector uses XML. This illustration shows the COM connector architecture for guaranteed events using JD Edwards EnterpriseOne Tools 8.95 and later Tools releases:



COM connector architecture-guaranteed event delivery for JD Edwards EnterpriseOne Tools release 8.95 and later Tools releases

Note. You should have a basic understanding of the COM+ Events Service.

COM+ events supports Z events, real-time events, and XAPI events. COM+ Events Service is not dependent on JD Edwards EnterpriseOne setup for event generation.

See Also

Microsoft MSDN, http://www.msdn.microsoft.com/

JD Edwards EnterpriseOne Tools 8.96 Interoperability Guide, "Using Guaranteed Events"

JD Edwards EnterpriseOne Tools 8.96 Interoperability Guide, "Using Real-Time Events - Guaranteed"

JD Edwards EnterpriseOne Tools 8.96 Interoperability Guide, "Using XAPI Events - Guaranteed"

Setting Up the COM Connector for Guaranteed Events - 8.94

This section provides an overview of the process for setting up the COM connector to receive guaranteed events when you use JD Edwards EnterpriseOne Tools 8.94 with JD Edwards EnterpriseOne Applications 8.11 and discusses how to:

- Install and set up the COM connector for guaranteed events - 8.94
- Register components for the COM connector for guaranteed events - 8.94
- Subscribe to guaranteed events - 8.94
- Log COM guaranteed events - 8.94

Understanding COM Connector Set Up for Guaranteed Events - 8.94

You can install the COM connector so that you can receive guaranteed events using JD Edwards EnterpriseOne Tools 8.94 with JD Edwards EnterpriseOne Applications 8.11. Setting up the COM connector includes setting up security and setting up the identity as an interactive user. After you install and set up the COM connector, you set up a DCOM server on a JD Edwards EnterpriseOne server machine. DCOM enables COM objects in a distributed environment. To ensure that the interoperability client works properly, you must set up DCOM for both a server environment and for a client environment. You also register the COM connector components, subscribe to events, and log errors and messages.

Installing and Setting Up the COM Connector for Guaranteed Events - 8.94

Use these steps to install and set up the COM connector so that you can receive guaranteed event using JD Edwards EnterpriseOne Tools 8.94 with JD Edwards EnterpriseOne Applications 8.11.

Note. All of the COM connector required files will be installed with the JD Edwards EnterpriseOne client. If you have the JD Edwards EnterpriseOne client, ignore Step 1 and start with Step 2. If you do not have the JD Edwards EnterpriseOne client and you want to set up the COM connector on a third-party machine, start with Step 1.

---

1. Copy these files from the JD Edwards EnterpriseOne server (system\bin32) to a directory on the desired machine.

   For example, copy the files in c:\program files\JD Edwards to a non-JD Edwards EnterpriseOne client machine.

   - JDECOMConnector2.exe
   - JDECOMMN.dll
   - callobject.dll
   - comlog.dll
   - EventManager.dll
   - OneWorldInterfaceTx.dll
   - xmlinterop.dll
   - jdel.dll
   - jdethread.dll
   - jdeunicode.dll
   - ustdio.dll
   - icuil8n.dll
   - jdeinterop.ini to c:\(root directory)
   - checkver.exe
   - ICUUC.dll
   - Icu\data*.*
   - IXXML4C2_3.dll

- EventClass.dll
- EventListener.dll
- JMSEvent.dll (8.94 only)

2. Create a new directory Icu\data\ on the machine where the COM server is located.

   Copy all of the files from the JD Edwards EnterpriseOne server in folder system\Locale\xml*.* into Icu\data\. Create a new system variable, ICU_DATA, in the environment variables of the system properties and specify the path to the Icu\data\ as the value.

3. Add JVM.dll path from IBM JDK1.4 in the path variable of the environment variables.

   This is required only for 8.94.

4. Use these steps to register the COM connector:

   a. Run this command:

      ```
      c:\programfiles\JDEdwards\JDECOMConnector2.exe /RegServer
      ```

   b. Go to c:\programfiles\JDEdwards\ Or c:\b9\system\bin32 and run these commands:

      ```
      regsvr32 EventManager.dll
      regsvr32 EventClass.dll
      ```

5. Create the JDEinterop.ini file by setting the JD Edwards EnterpriseOne server and port values to the JD Edwards EnterpriseOne application server with which you want the COM server to communicate.

   The COM server is now ready.

6. Use these steps to set up security on the COM server:

   a. From the Start menu, select Run.

   b. Enter Dcomcnfg.exe.

   c. On Distributed COM Configuration Properties, click the Default Security tab.

   d. Click the Edit Default Button in Default Access Permissions group.

   e. The Registry Value Permissions form appears. Some entries might already be present.

   f. On Registry Value Permissions, click Add.

   g. On Add Users and Groups, select the appropriate domain from the List Names From option.

   h. Click Everyone, and then click Add.Type of access should be Allow Access.

   i. Click OK.

   No setup is required for default configuration permissions.

7. Use these steps to set up the identity as interactive user:

   a. Run DCOMCnfg.

   b. On Distributed COM Configuration Properties, select JDECOMConnector2, and then click Properties.

   c. On JDECOMConnector2Properties, click the Identity tab, and then select the interactive user option.

   d. Click Apply to apply the change.

Note. Every time you register the connector, you must set up the identity as an interactive user. If you copy the JDECOMConnector2.exe using Explorer, Explorer reruns the registration, and you must set up the identity as an interactive user.

To use Callbacks (Connection Points) with the COM solution, repeat these steps for setting up the identity as an interactive user on the COM client machine. Most of the shipped examples use Callbacks and require that you open the security on the client machine.

8. Use these steps to set up DCOM for a client environment:
    a. From a DOS prompt on the DCOM client machine, run jdecomconnector2.exe /RegServer.
    b. At the prompt, enter oleview.exe.
    c. From the menu bar, select oleview.
    d. Click View and select Expert Mode.
    e. In the oleview window under Object Classes, double-click All Objects, and wait for all objects to appear.
    f. Under All Objects, find and click Connector Class.
    g. Click the Implementation tab on the right-side panel, and then click the local server and remove anything that appears in the editing window.
    h. On the Activation tab, select the Launch as Interactive User option.
    i. In Remote Machine Name, enter the COM server machine name.

Repeat steps 5 through 8 for MathNumeric Class.Start the DCOM client application.

Start the DCOM client application.

Registering Components for COM Connector - 8.94

So that subscribers can find an event class and subscribe to it, the JD Edwards EnterpriseOne event class must be registered with COM+. In addition, COM+ requires a type library that describes the event interface and methods so that subscribers and publishers can be properly matched and connected. The type library must reside in or be accompanied by a self-registering DLL.

To register the JD Edwards EnterpriseOne Events Class with COM+ Services, you must:

- Add a new COM+ application for the JD Edwards EnterpriseOne event class.
- Install the JD Edwards EnterpriseOne event class.

Note. Before you register the JD Edwards EnterpriseOne Event Class with COM+ Services, set up the COM server. The COM server can be set up on either a JD Edwards EnterpriseOne machine or a non-JD Edwards EnterpriseOne machine (third-party machine), or both.

See Also

Chapter 4, “Deploying the COM Solution for Business Function Execution,” Installing COM Connector, page 24

Subscribing to Events - 8.94

The COM connector supports event subscriptions from JD Edwards EnterpriseOne (JD Edwards EnterpriseOne server and Transaction server). The COM connector connects to the JD Edwards EnterpriseOne Transaction server to receive its subscribed events.

You must set up the jdeinterop.ini file, including the [JMSEVENTS] section. Also, you must add the path (not including the file name) to the appropriate jvm.dll file in the system's path environment variable. For connecting to a JD Edwards EnterpriseOne Transaction server running in WebSphere, you must use the jvm.dll provided by WebSphere.

Logging COM Events - 8.94

Logging for COM events is entered in the interopDebug.log file. The error log is interop.log.

Setting Up the COM Connector for Guaranteed Events - 8.95

This section provides an overview of the process for setting up the COM connector to receive guaranteed events when you use JD Edwards EnterpriseOne Tools 8.95 and later Tools releases with JD Edwards EnterpriseOne Applications 8.9 and later releases.

Understanding COM Connector Setup for Guaranteed Events - 8.95

You can install the COM connector so that you can receive guaranteed events using JD Edwards EnterpriseOne Tools 8.95 and later Tools releases with JD Edwards EnterpriseOne Applications 8.9 and later Applications releases. Setting up the COM connector includes setting up security and setting up the identity as an interactive user. After you install and set up the COM connector, you set up a DCOM server on a JD Edwards EnterpriseOne server machine. DCOM enables COM objects in a distributed environment. To ensure that the interoperability client works properly, you must set up DCOM for both a server environment and for a client environment. You also register the COM connector components, subscribe to events, and log errors and messages.

Installing and Setting Up the COM Connector for Guaranteed Events - 8.95

Use these steps to install and set up the COM connector so that you can receive guaranteed event using JD Edwards EnterpriseOne Tools 8.95 and later Tools releases with JD Edwards EnterpriseOne Applications 8.9 and later Applications releases.

Note. All of the COM connector required files will be installed with the JD Edwards EnterpriseOne client. If you have the JD Edwards EnterpriseOne client, ignore Step 1 and start with Step 2. If you do not have the JD Edwards EnterpriseOne client and you want to set up the COM connector on a third-party machine, start with Step 1.

---

1. Copy these files from the JD Edwards EnterpriseOne server (system\bin32) to a directory on the desired machine. For example, copy the files in c:\program files\JDEdwards to a non-JD Edwards EnterpriseOne client machine.

   - JDECOMConnector2.exe

- JDECOMMN.dll
- callobject.dll
- comlog.dll
- EventManager.dll
- OneWorldInterfaceTx.dll
- xmlinterop.dll
- jdel.dll
- jdethread.dll
- jdeunicode.dll
- ustdio.dll
- icuil8n.dll
- jdeinterop.ini to c:\(root directory)
- checkver.exe
- ICUUC.dll
- Icu\data*.*
- IXXML4C2_3.dll
- EventClass.dll
- EventListener.dll
- EventHandler.dll
- ClientService.dll

2. Create a new directory Icu\data\ on the machine where the COM server is located.

   Copy all of the files from the JD Edwards EnterpriseOne server in folder system\Locale\xml*.* into Icu\data\. Create a new system variable, ICU_DATA, in the environment variables of the system properties and specify the path to the Icu\data\ as the value.

3. Use these steps to register the COM Connector:

   a. Run this command:

      ```
      c:\programfiles\JDEdwards\JDECOMConnector2.exe /RegServer
      ```

   b. Go to c:\programfiles\JDEdwards\ Or c:\b9\system\bin32 and run these commands:

      ```
      regsvr32 EventManager.dll
      regsvr32 EventClass.dll
      ```

4. Create the JDEinterop.ini file by setting the JD Edwards EnterpriseOne server and port values to the JD Edwards EnterpriseOne application server with which you want the COM server to communicate.

   The COM server is now ready.

5. Use these steps to set up security on the COM server:

   a. From the Start menu, select Run.

   b. Enter Dcomcnfg.exe.

   c. On Distributed COM Configuration Properties, click the Default Security tab.

d. Click the Edit Default Button in Default Access Permissions group.

e. The Registry Value Permissions form appears. Some entries might already be present.

f. On Registry Value Permissions, click Add.

g. On Add Users and Groups, select the appropriate domain from the List Names From option.

h. Click Everyone, and then click Add.Type of access should be Allow Access.

i. Click OK.

No setup is required for default configuration permissions.

6. Use these steps to set up the identity as an interactive user:

   a. Run DCOMCnfg.

   b. On Distributed COM Configuration Properties, select JDECOMConnector2, and then click Properties.

   c. On JDECOMConnector2Properties, click the Identity tab, and then select the interactive user option.

   d. Click Apply to apply the change.

   Note. Every time you register the connector, you must set up the identity as an interactive user. If you copy the JDECOMConnector2.exe using Explorer, Explorer reruns the registration, and you must set up the identity as an interactive user.

   To use Callbacks (Connection Points) with the COM solution, repeat these steps for setting up the identity as an interactive user on the COM client machine. Most of the shipped examples use Callbacks and require that you open the security on the client machine.

7. Use these steps to set up DCOM for a client environment:

   - From a DOS prompt on the DCOM client machine, run jdecomconnector2.exe /RegServer.
   - At the prompt, enter oleview.exe.
   - From the menu bar, select oleview.
   - Click View and select Expert Mode.
   - In the oleview window under Object Classes, double-click All Objects, and wait for all objects to appear.
   - Under All Objects, find and click Connector Class.
   - Click the Implementation tab on the right-side panel, and then click the local server and remove anything that appears in the editing window.
   - On the Activation tab, select the Launch as Interactive User option.
   - In Remote Machine Name, enter the COM server machine name.

Repeat steps 5 through 8 for MathNumeric Class.Start the DCOM client application.

Start the DCOM client application.

Registering Components for COM Connector - 8.95

So that subscribers can find an event class and subscribe to it, the JD Edwards EnterpriseOne event class must be registered with COM+. In addition, COM+ requires a type library that describes the event interface and methods so that subscribers and publishers can be properly matched and connected. The type library must reside in or be accompanied by a self-registering DLL.

To register the JD Edwards EnterpriseOne Events Class with COM+ Services, you must:

- Add a new COM+ application for the JD Edwards EnterpriseOne event class.
- Install the JD Edwards EnterpriseOne event class.

Note. Before you register the JD Edwards EnterpriseOne Event Class with COM+ Services, set up the COM server. The COM server can be set up on either a JD Edwards EnterpriseOne machine or a non-JD Edwards EnterpriseOne machine (third-party machine), or both.

See Also

Chapter 4, "Deploying the COM Solution for Business Function Execution," Installing COM Connector, page 24

Subscribing to Events - 8.95

The COM connector supports event subscriptions from JD Edwards EnterpriseOne (JD Edwards EnterpriseOne server and Transaction server). The COM connector connects to the JD Edwards EnterpriseOne Transaction server to receive its subscribed events.

Logging COM Events - 8.95

Logging for COM events is entered in the interopDebug.log file. The error log is interop.log.

Implementing JD Edwards EnterpriseOne Interfaces

This section provides an overview about implementing the JD Edwards EnterpriseOne interface and discusses how to:

- Create a COM+ component.
- Log on to the COM connector.
- Subscribe to an event.
- Integrate with BizTalk.
- Add a new application.
- Install the event class.

Implementing a JD Edwards EnterpriseOne Interface

You must develop an object that implements the IOWEvent interface. For further discussion and for code samples in this document, the name EventSink is used as the object name. The object that you develop to implement the IOWEvent can have a different name. EventSink implements the IOWEvent interface and the method within the interface, and then consumes the JD Edwards EnterpriseOne event. The EventSink implementation is client specific. EventSink receives the event from JD Edwards EnterpriseOne by implementing the interface specified in EventClass.

This code outline shows how to develop an EventSink component:


```
Option Explicit
Implements IOWEvent
Public Event OneWorldEvent(ByVal EventName As String, ByVal Data As String)

Public Sub IOWEvent_OneWorldEvent(ByVal EventName As String, ByVal Data
As String)
'// Add code specific to the client implementation here
    RaiseEvent OneWorldEvent(EventName, Data)
End Sub
```

This list outlines the steps for you to follow to use the EventManager library and MessageHandler Interface to subscribe to events.

1. Log on to the connector. Successful logon returns an access number.
2. Create the EventSink object.
3. Create the MessageHandler object.
4. Call methods on the MessageHandle for Subscribe, Unsubscribe, GetTemplate, and GetEventList for the respective event.
5. To keep the session alive and not time out from receiving events, call the UpdateOutBoundSessionTime method on the connector interface.

   This method updates the user session time to the current time.
6. To subscribe to the events as persistent, register VB EventSink in the COM+ Component Services and add the subscription for the EventClass.

Creating a COM+ Component

This sample code is for creating a COM+ component named EventSink.dll. EventSink implements the EventClass interface IOWEvent(). You can use a name other than EventSink.

EventSink: OneWorldTransientEventSink.cls

This code illustrates how to create a COM+ component:

```
Option Strict Off
Option Explicit On
<System.Runtime.InteropServices.ProgId
("OneWorldTransientEventSink_NET.OneWorldTransientEventSink")>
Public Class OneWorldTransientEventSink
     Implements EventClass.IOWEvent

     Public Event OneWorldEvent(ByVal EventName As String, ByVal
Data As String)

     Public Sub IOWEvent_OneWorldEvent(ByVal EventName As String,
ByVal Data As String) Implements EventClass.IOWEvent.OneWorldEvent
          Dim flsObject As New Scripting.FileSystemObject
          Dim varEventFile As Scripting.TextStream
          Dim strEventFile As String
          strEventFile = "C:\temp\eventDataPer.xml"
          'UPGRADE_WARNING: Dir has a new behavior. Click for more:
```



```
'ms-help://MS.VSCC.2003/commoner/redir/redirect.htm?keyword=
"vbup1041"'
          If Dir(strEventFile) = "" Then
               varEventFile = flsObject.CreateTextFile(strEventFile,
False, False)
          Else
               varEventFile = flsObject.OpenTextFile(strEventFile,
Scripting.IOMode.ForWriting, False)
          End If

          varEventFile.WriteLine(Data)
          varEventFile.Close()
          RaiseEvent OneWorldEvent(EventName, Data)
     End Sub
End Class
```

Logging on to the COM Connector

This sample code logs on to the COM connector, creates the MessageHandler object, and performs Subscribe, Unsubscribe, GetTemplate, and GetList. Before executing the subscriber, use the Regsvr32 command to register COMConnector.dll.

COMConnector: frmLogin.frm

This code sample shows logging on to the COM connector:

```
Option Strict Off
Option Explicit On

Friend Class frmLogin
   Inherits System.Windows.Forms.Form

   Public bLoginEnv As Boolean

   Private Sub cmdCancel_Click(ByVal eventSender As System.Object,
ByVal eventArgs As System.EventArgs) Handles cmdCancel.Click
          'set the global var to false
          'to denote a failed login
          bLoginEnv = False
          Me.Hide()
   End Sub

   Private Sub cmdOK_Click(ByVal eventSender As System.Object,
ByVal eventArgs As System.EventArgs) Handles cmdOK.Click
          'check for correct password
          If txtUserName.Text = "" Or txtenvironment.Text = "" Then
                 bLoginEnv = False
                 MsgBox("Must Enter User Name and Environment to
continue")
          Else
                 bLoginEnv = True
```

```
                Me.Hide()
            End If
    End Sub
End Class
```

COMConnector Common.bas

This code sample shows creating the message handler:

```
Option Strict Off
Option Explicit On
Module Common
    Dim conn As New JDECOMCONNECTOR2Lib.Connector
    Dim connRole As JDECOMCONNECTOR2Lib.IConnector2
    'Dim messageHandler As New messageHandler
    'Dim mHandlerInterface As ImessageHandler
    Dim lngAccessNumber As Integer
    Public Sub comm_Initialize()
        connRole = conn
        On Error GoTo errorHandler
        frmLogin.DefInstance.bLoginEnv = False
        frmLogin.DefInstance.Show()
        While Not frmLogin.DefInstance.bLoginEnv
            System.Windows.Forms.Application.DoEvents()
        End While
        lngAccessNumber = connRole.E1_Event_Login(frmLogin.
DefInstance.
txtUserName.Text, frmLogin.DefInstance.txtPassword.Text, frmLogin.
DefInstance.txtenvironment.Text, frmLogin.DefInstance.txtrole.Text)
        'Debugging  Purpose
        'lngAccessNumber = connRole.E1_Event_Login("JP6849777",
"PASSWORD", "TDEVNIS2", "*ALL")
        connRole = conn
        Exit Sub
errorHandler:
        MsgBox("Login Failed. You can't Use this Application")

    End Sub

    ' NOTE: the code in this module is particular to this prototype.
    ' Different code is used in a production version to send messages to
    ' JD Edwards EnterpriseOne using JD Edwards communication protocols.

    Public Sub SendSubscriptionToOneWorld(ByRef eventName As String,
ByRef oneworldevent As EventClass.IOWEvent, ByRef mode As Integer)
        'mHandlerInterface.SubscribeEvent lngAccessNumber, conn,
eventName, oneworldevent, mode
        On Error GoTo errorHandler
        connRole.E1_Event_Subscribe(lngAccessNumber, oneworldevent)
        Exit Sub
errorHandler:
```



```
        MsgBox("Subscirbe Method Failed. You can't Use this
Application")
    End Sub
    Public Sub SendUnSubscribeToOneWorld(ByRef eventName As String,
ByRef oneworldevent As EventClass.IOWEvent, ByRef mode As Integer)
        On Error GoTo errorHandler
        'mHandlerInterface.UnSubscribeEvent lngAccessNumber, conn,
eventName, oneworldevent, mode
        connRole.E1_Event_UnSubscribe(lngAccessNumber)
        Exit Sub
errorHandler:
        MsgBox("UnSubscirbe Method Failed. You can't Use this
Application")
    End Sub
    Public Sub SendLogoffToOneWorld()
        'mHandlerInterface.SubscribeEvent lngAccessNumber, conn,
eventName, oneworldevent, mode
        On Error GoTo errorHandler
        connRole.E1_Event_Logoff(lngAccessNumber)
        Exit Sub
errorHandler:
        MsgBox("LogOff Method Failed. Terminate ComConnector
Process and End the Application")
    End Sub
    Public Sub getEventListFromOneWorld(ByRef eventList As String)
        On Error GoTo errorHandler
        'mHandlerInterface.GetEventList lngAccessNumber, conn,
eventList
        eventList = connRole.E1_Event_GetEventList(lngAccessNumber)
        Exit Sub
errorHandler:
        MsgBox("GetEventList Method Failed. You can't Use this
Application")
    End Sub
    Public Sub getEventTemplateFromOneWorld(ByRef eventName As
String, ByRef eventTemplate As String)
        On Error GoTo errorHandler
        'mHandlerInterface.GetEventTemplate lngAccessNumber,
eventName, conn, eventTemplate
        Exit Sub
errorHandler:
        MsgBox("GetEventTemplate Method Failed. You can't Use this
Application")
    End Sub
End Module
```

COMConnector: SubscriptionManager

This code sample shows event subscription and unsubscribe:

```
Option Strict Off
```



```
Option Explicit On
<System.Runtime.InteropServices.ProgId("SubscriptionManager_NET.
SubscriptionManager")> Public Class SubscriptionManager

   'Private Const m_OneWorldEventCLSID = "{1E645180-6C93-4704-85C6-
57775E2ED2FC}"
   Private m_SubscribedEvents As Collection

   'UPGRADE_NOTE: Class_Initialize was upgraded to Class_Initialize_
Renamed. Click for more: 'ms-help://MS.VSCC.2003/commoner/redir/
redirect.htm?keyword="vbup1061"'
   Private Sub Class_Initialize_Renamed()
          m_SubscribedEvents = New Collection
          comm_Initialize()
   End Sub
   Public Sub New()
          MyBase.New()
          Class_Initialize_Renamed()
   End Sub
   Public Sub GetEventList(ByRef eventList As String)
          getEventListFromOneWorld(eventList)
   End Sub

   Public Sub Logoff()
          SendLogoffToOneWorld()
   End Sub

   Public Sub CreateTransientSubscription(ByRef eventName As String,
ByRef oneworldevent As EventClass.IOWEvent)
          SubscribeToOneWorldEvent(eventName, oneworldevent, 0)
   End Sub
   Public Sub CreatePersistentSubscription(ByRef eventName As
String, ByRef oneworldevent As EventClass.IOWEvent)
          SubscribeToOneWorldEvent(eventName, oneworldevent, 1)
   End Sub
   Public Sub RemoveTransientSubscription(ByRef eventName As String,
ByRef oneworldevent As EventClass.IOWEvent)
          UnSubscribeToOneWorldEvent(eventName, oneworldevent, 0)
   End Sub
   Public Sub RemovePersistentSubscription(ByRef eventName As
String, ByRef oneworldevent As EventClass.IOWEvent)
          UnSubscribeToOneWorldEvent(eventName, oneworldevent, 1)
   End Sub
   Public Sub GetEventTemplate(ByRef eventName As String, ByRef
eventTemplate As String)
          getEventTemplateFromOneWorld(eventName, eventTemplate)
   End Sub
   Public Sub SubscribeToOneWorldEvent(ByRef eventName As String,
ByRef  oneworldevent As EventClass.IOWEvent, ByRef mode As Integer)
          'Private Function SubscribeToOneWorldEvent(EventName As
```



```
 String) As Boolean
          ' we've already subscribed if the subscription is in our
list
          Dim alreadySubscribed As Boolean
          'UPGRADE_WARNING: Couldn't resolve default property of
object CollectionContainsString(). Click for more: 'ms-help:
//MS.VSCC.2003/commoner/redir/redirect.htm?keyword="vbup1037"'
          alreadySubscribed = (CollectionContainsString
(m_SubscribedEvents, eventName) = True)

          ' now do the right thing...
          If (alreadySubscribed = False) Then
                 ' this instance of the COMConnector has not seen this
                 ' event before, so add it to our list...
                 m_SubscribedEvents.Add((eventName))

                 ' ...and go ahead and subscribe to the event from
JD Edwards EnterpriseOne
                 SendSubscriptionToOneWorld(eventName,
oneworldevent, mode)
          End If

          'SubscribeToOneWorldEvent = alreadySubscribed
   End Sub

   'UPGRADE_NOTE: str was upgraded to str_Renamed. Click for more:
'ms-help://MS.VSCC.2003/commoner/redir/redirect.htm?keyword="vbup1061"'
   Private Function CollectionContainsString(ByRef col As
Collection, ByRef str_Renamed As String) As Object
          Dim colItem As Object
          For    Each colItem In col
                 'UPGRADE_WARNING: Couldn't resolve default
property of object colItem. Click for more: 'ms-help:
//MS.VSCC.2003/commoner/redir/redirect.htm?keyword="vbup1037"'
                 If (colItem = str_Renamed) Then
                        'UPGRADE_WARNING: Couldn't resolve default
property of object CollectionContainsString. Click for more:
'ms-help://MS.VSCC.2003/commoner/redir/redirect.htm?keyword="vbup1037"'
                        CollectionContainsString = True
                        Exit Function
                 End If
          Next colItem
          'UPGRADE_WARNING: Couldn't resolve default property of
object CollectionContainsString. Click for more:
'ms-help://MS.VSCC.2003/commoner/redir/redirect.htm?keyword="vbup1037"'
          CollectionContainsString = False
   End Function

      Public Sub UnSubscribeToOneWorldEvent(ByRef eventName As String,
ByRef oneworldevent As EventClass.IOWEvent, ByRef mode As Integer)
```

```
          Dim alreadySubscribed As Boolean
          'alreadySubscribed = (CollectionContainsString
(m_SubscribedEvents.Item, eventName))

          ' now do the right thing...
          'If (alreadySubscribed = True) Then
          ' this instance of the COMConnector has not seen this

event before, so
          ' remove it from the list...
          alreadySubscribed = (RemoveFromCollection
(m_SubscribedEvents, eventName))
          If (alreadySubscribed = False) Then
                MsgBox("Event Not Subscribed")
          Else

                'm_SubscribedEvents.Remove ()

                ' ...and go ahead and subscribe to the event from
JD Edwards EnterpriseOne
                SendUnSubscribeToOneWorld(eventName, oneworldevent,
mode)
          End If
          '  End If
   End Sub
   'UPGRADE_NOTE: str was upgraded to str_Renamed. Click for more:
'ms-help://MS.VSCC.2003/commoner/redir/redirect.htm?keyword="vbup1061"'
   Private Function RemoveFromCollection(ByRef col As Collection,
ByRef str_Renamed As String) As Object
          Dim colItem As Object
          Dim count As Short
          count = 0
          For    Each colItem In col
                count = count + 1
                'UPGRADE_WARNING: Couldn't resolve default
property of object colItem. Click for more: 'ms-help:
//MS.VSCC.2003/commoner/redir/redirect.htm?keyword="vbup1037"'
                If (colItem = str_Renamed) Then
                      col.Remove(count)
                      'UPGRADE_WARNING: Couldn't resolve default
property of object RemoveFromCollection. Click for more:
'ms-help://MS.VSCC.2003/commoner/redir/redirect.htm?keyword="vbup1037"'
                      RemoveFromCollection = True
                      Exit Function
                End If
          Next colItem
          'UPGRADE_WARNING: Couldn't resolve default property of
object RemoveFromCollection. Click for more: 'ms-help:
//MS.VSCC.2003/commoner/redir/redirect.htm?keyword="vbup1037"'
          RemoveFromCollection = False
```



```
   End Function
End Class
```

Subscribing to an Event

Subscriber is the GUI that gets the EventsList, EventTemplate, Subscribe, and Unsubscribe. Subscriber is built as a VB executable. Typical usage is to get the EventList first, which populates the list of options with the events that are supported by the JD Edwards EnterpriseOne server. Select the event that needs to be subscribed from the JD Edwards EnterpriseOne server and the type of subscription. Click Subscribe to add a Subscription, or click Unsubscribe to unsubscribe from the JD Edwards EnterpriseOne server. The Subscribed events and the Received events are in separate boxes. The received event is displayed in the window on the right. The event received can be integrated with BizTalk by choosing the Enable BizTalk Integration option. You should have previously set up BizTalk; if not already installed, install the BizTalk Server 2000 Developer. If the Module 1 tutorial in the BizTalk Server documentation runs properly, then the BizTalk Server is properly installed. Before building the subscriber, you should use the Regsvr32 command to register EventSink.dll and COMConnector.dll.

Subscriber: MainForm.frm

This code sample is for the GUI and the control buttons on the GUI. This code should be built along with the BizTalk.cls, after registering the COMConnector.dll and MyEventSink.dll.

```
VERSION 5.00
Object = "{EAB22AC0-30C1-11CF-A7EB-0000C05BAE0B}#1.1#0"; "shdocvw.dll"
Object = "{831FDD16-0C5C-11D2-A9FC-0000F8754DA1}#2.0#0"; "mscomctl.ocx"
Begin VB.Form MainForm
   Caption         =   "Subscriber Client"
   ClientHeight    =   7470
   ClientLeft      =   3555
   ClientTop       =   2820
   ClientWidth     =   11655
   LinkTopic       =   "Form1"
   ScaleHeight     =   7470
   ScaleWidth      =   11655
   Begin VB.Frame grpSubscribedEvents
      Caption         =   "Subscribed Events"
      Height          =   2895
      Index           =   1
      Left            =   120
      TabIndex        =   17
      Top             =   2160
      Width           =   2775
      Begin VB.CommandButton Command1
         Caption         =   "Clear"
         Height          =   375
         Left            =   4560
         TabIndex        =   18
         Top             =   2280
         Width           =   975
      End
      Begin MSComctlLib.ListView lvwSubscribedEvents
         Height          =   1695
```

```
         Left            =   120
         TabIndex        =   19
         Top             =   360
         Width           =   2535
         _ExtentX        =   4471
         _ExtentY        =   2990
         View            =   2
         LabelWrap       =   -1  'True
         HideSelection   =   -1  'True
         _Version        =   393217
         ForeColor       =   -2147483640
         BackColor       =   -2147483643
         BorderStyle     =   1
         Appearance      =   1
         NumItems        =   2
         BeginProperty ColumnHeader(1) {BDD1F052-858B-11D1-B16A-
00C0F0283628}
            Key             =   "colEventName"
            Text            =   "Event Name"
            Object.Width           =   2540
         EndProperty
         BeginProperty ColumnHeader(2) {BDD1F052-858B-11D1-B16A-
00C0F0283628}
            SubItemIndex    =   1
            Key             =   "colData"
            Text            =   "Data"
            Object.Width           =   6174
         EndProperty
      End
   End
   Begin VB.CommandButton btnGetEventTemplate
      Caption         =   "Get Template"
      Height          =   375
      Left            =   3720
      TabIndex        =   14
      Top             =   120
      Width           =   1455
   End
   Begin VB.CommandButton btnGetEventList
      Caption         =   "Get Event List"
      Height          =   375
      Left            =   600
      TabIndex        =   13
      Top             =   120
      Width           =   1455
   End
   Begin SHDocVwCtl.WebBrowser wbEventData
      Height          =   6375
      Left            =   6240
      TabIndex        =   12
```



```
      Top             =   360
      Width           =   5175
      ExtentX         =   9128
      ExtentY         =   11245
      ViewMode        =   0
      Offline         =   0
      Silent          =   0
      RegisterAsBrowser=   0
      RegisterAsDropTarget=   1
      AutoArrange     =   0   'False
      NoClientEdge    =   0   'False
      AlignLeft       =   0   'False
      NoWebView       =   0   'False
      HideFileNames   =   0   'False
      SingleClick     =   0   'False
      SingleSelection =   0   'False
      NoFolders       =   0   'False
      Transparent     =   0   'False
      ViewID          =   "{0057D0E0-3573-11CF-AE69-08002B2E1262}"
      Location        =   ""
   End
   Begin VB.CheckBox chkEnableBizTalkIntegration
      Caption         =   "Enable BizTalk Integration"
      Height          =   255
      Left            =   240
      TabIndex        =   8
      Top             =   5280
      Width           =   2535
   End
   Begin VB.Frame grpEnableBizTalkIntegration
      Height          =   975
      Left            =   120
      TabIndex        =   7
      Top             =   5640
      Width           =   5775
      Begin VB.TextBox txtScheduleFile
         Height          =   375
         Left            =   1440
         TabIndex        =   10
         Text            =   "sked:///\vbeventsdemo\Products\
VBCOMConnector\BizTalk\Buyer1.skx"
         Top             =   360
         Width           =   4095
      End
      Begin VB.Label lblScheduleFile
         Alignment       =   1  'Right Justify
         Caption         =   "Schedule File:"
         Height          =   255
         Left            =   240
         TabIndex        =   9
```

```
         Top             =   480
         Width           =   1095
      End
   End
   Begin VB.CommandButton btnClose
      Caption         =   "Close"
      Height          =   375
      Left            =   5760
      TabIndex        =   3
      Top             =   6960
      Width           =   975
   End
   Begin VB.Frame grpReceivedEvents
      Caption         =   "Received Events"
      Height          =   2895
      Index           =   0
      Left            =   3000
      TabIndex        =   6
      Top             =   2160
      Width           =   2895
      Begin VB.CommandButton btnClear
         Caption         =   "Clear"
         Height          =   375
         Index           =   0
         Left            =   1680
         TabIndex        =   2
         Top             =   2280
         Width           =   975
      End
      Begin MSComctlLib.ListView lvwReceivedEvents
         Height          =   1695
         Left            =   120
         TabIndex        =   1
         Top             =   360
         Width           =   2655
         _ExtentX        =   4683
         _ExtentY        =   2990
         View            =   2
         LabelWrap       =   -1  'True
         HideSelection   =   -1  'True
         _Version        =   393217
         ForeColor       =   -2147483640
         BackColor       =   -2147483643
         BorderStyle     =   1
         Appearance      =   1
         NumItems        =   2
         BeginProperty ColumnHeader(1) {BDD1F052-858B-11D1-B16A-
00C0F0283628}
            Key             =   "colEventName"
            Text            =   "Event Name"
```



```
            Object.Width           =   2540
         EndProperty
         BeginProperty ColumnHeader(2) {BDD1F052-858B-11D1-B16A-
00C0F0283628}
            SubItemIndex    =   1
            Key             =   "colData"
            Text            =   "Data"
            Object.Width           =   6174
         EndProperty
      End
   End
   Begin VB.Frame grpSubscriptions
      Caption         =   "Subscriptions"
      Height          =   1215
      Left            =   120
      TabIndex        =   4
      Top             =   720
      Width           =   5775
      Begin VB.CheckBox chkPersist
         Caption         =   "Persist"
         Height          =   255
         Left            =   1560
         TabIndex        =   16
         Top             =   840
         Width           =   975
      End
      Begin VB.ComboBox cEventList
         Height          =   315
         Left            =   1560
         Sorted          =   -1  'True
         TabIndex        =   15
         Top             =   360
         Width           =   2295
      End
      Begin VB.CommandButton btnUnsubscribe
         Caption         =   "UnSubscribe"
         Height          =   375
         Left            =   4200
         TabIndex        =   11
         Top             =   720
         Width           =   1095
      End
      Begin VB.CommandButton btnSubscribe
         Caption         =   "Subscribe"
         Height          =   375
         Left            =   4200
         TabIndex        =   0
         Top             =   240
         Width           =   1095
      End
```

```
      Begin VB.Label lblEventName
         Alignment       =   1  'Right Justify
         Caption         =   "Event Name:"
         Height          =   255
         Left            =   360
         TabIndex        =   5
         Top             =   360
         Width           =   1095
      End
   End
End
Attribute VB_Name = "MainForm"
Attribute VB_GlobalNameSpace = False
Attribute VB_Creatable = False
Attribute VB_PredeclaredId = True
Attribute VB_Exposed = False
Option Explicit

' ------------------------------- ** ---------------------------------
'                              Member Variables
' ------------------------------- ** ---------------------------------

Private m_SubscriptionManager As SubscriptionManager
Private WithEvents m_OneWorldTransientEventSink As
OneWorldTransientEventSink
Attribute m_OneWorldTransientEventSink.VB_VarHelpID = -1
Private Sub Combo1_Change()

End Sub
Private Sub Check1_Click()

End Sub

Private Sub btnClear_Click(Index As Integer)
    lvwReceivedEvents.ListItems.Clear
End Sub

'----------------------------- ** ---------------------------------
'                           GetEventTemplate
'----------------------------- ** ---------------------------------
Private Sub btnGetEventTemplate_Click()
    Dim EventName As String
    Dim EventTemplate As String
    EventName = cEventList.List(cEventList.ListIndex)
    'm_SubscriptionManager.GetEventTemplate EventName, EventTemplate
    Dim flsObject As New Scripting.FileSystemObject
    Dim varTemplateFile As TextStream
    Dim strTemplateFile As String
    strTemplateFile = "C:\temp\event_template.xml"
    If Dir(strTemplateFile) = "" Then
```



```
        Set varTemplateFile = flsObject.CreateTextFile
(strTemplateFile, False, False)
    Else
        Set varTemplateFile = flsObject.OpenTextFile
(strTemplateFile,ForWriting, False)
    End If

    varTemplateFile.WriteLine EventTemplate
    varTemplateFile.Close

    wbEventData.Navigate "c:\temp\event_template.xml"
End Sub

' -------------------------------- ** --------------------------------
'                                Event Handlers
' -------------------------------- ** --------------------------------

Private Sub Form_Load()
    Set m_SubscriptionManager = New SubscriptionManager
    Set m_OneWorldTransientEventSink = New OneWorldTransientEventSink

    'EnableBizTalkIntegrationGroup
End Sub

Private Sub m_OneWorldTransientEventSink_OneWorldEvent(ByVal EventName
As String, ByVal Data As String)
    ' add the event name and payload to the list
    Dim mTempItem As ListItem
    Set mTempItem = lvwReceivedEvents.ListItems.Add()
    mTempItem.Text = EventName
    'mTempItem.SubItems(1) = Data
    Dim flsObject As New Scripting.FileSystemObject
    Dim varEventFile As TextStream
    Dim strEventFile As String
    strEventFile = "C:\temp\eventData.xml"
    If Dir(strEventFile) = "" Then
        Set varEventFile = flsObject.CreateTextFile(strEventFile,
False, False)
    Else
        Set varEventFile = flsObject.OpenTextFile(strEventFile,
ForWriting, False)
    End If

    varEventFile.WriteLine Data
    varEventFile.Close
    wbEventData.Navigate "c:\temp\eventdata.xml"

    ' send the event to BizTalk (if it is enabled)
    'If (chkEnableBizTalkIntegration.Value = Checked) Then
        'Dim oBizTalk As BizTalk
```

```
        'Set oBizTalk = New BizTalk
        'oBizTalk.RunSchedule txtScheduleFile.Text, Data
   ' End If
End Sub

'------------------------------ ** ----------------------------------
'                           GetEventList
'------------------------------ ** ----------------------------------
Private Sub btnGetEventList_Click()
    Dim events As String
    Dim myValue As String
    Dim myString As String
    Set m_SubscriptionManager = New SubscriptionManager
    m_SubscriptionManager.GetEventList events

    cEventList.Clear
    events = "RTSOOUT"
    myString = events
    'Do Until events = ""
        'If InStr(1, myString, ":") > 0 Then
        '    myValue = Left(myString, InStr(1, myString, ":") - 1)
        '    myString = Mid(myString, InStr(1, myString, ":") + 1)
        'Else
        '    myValue = myString
        '    events = ""
        'End If

        'cEventList.AddItem myValue
   ' Loop
   cEventList.AddItem myString
    cEventList.ListIndex = 0
End Sub

'------------------------------ ** ----------------------------------
'                           Subscribe Event
'------------------------------ ** ----------------------------------
Private Sub btnSubscribe_Click()
    ' subscribe to the named event.
    Dim EventName As String
    EventName = cEventList.List(cEventList.ListIndex)
    If (chkPersist.Value = Checked) Then
       m_SubscriptionManager.CreatePersistentSubscription EventName,
m_OneWorldTransientEventSink
    Else
       m_SubscriptionManager.CreateTransientSubscription EventName,
m_OneWorldTransientEventSink
    End If
    Dim mTempItem As ListItem
    Set mTempItem = lvwSubscribedEvents.ListItems.Add()
    mTempItem.Text = EventName
```



```
End Sub

'---------------------------- ** ----------------------------------
'                            UnSubscribe Event
'---------------------------- ** ----------------------------------
Private Sub btnUnsubscribe_Click()
    Dim EventName As String
    EventName = cEventList.List(cEventList.ListIndex)
    Dim lstItem As ListItem
    Dim count As Integer
    Dim found As Boolean
    count = 0
    found = False
    For Each lstItem In lvwSubscribedEvents.ListItems
        count = count + 1
        If lstItem = EventName Then
            lvwSubscribedEvents.ListItems.remove (count)
            GoTo remove
            found = True
        End If
    Next
    If found = False Then
        MsgBox "Event Not Subscribed"
    End If
remove: If (chkPersist.Value = Checked) Then
       m_SubscriptionManager.RemovePersistentSubscription EventName,
m_OneWorldTransientEventSink
    Else
       m_SubscriptionManager.RemoveTransientSubscription EventName,
m_OneWorldTransientEventSink
    End If

End Sub

Private Sub chkEnableBizTalkIntegration_Click()
    'EnableBizTalkIntegrationGroup
End Sub
'---------------------------- ** ----------------------------------
'                    Clear the Received Events List
'---------------------------- ** ----------------------------------
Private Sub btnClear0_Click()
    ' clear the events from the list
    lvwReceivedEvents.ListItems.Clear
End Sub

Private Sub btnClose_Click()
    m_SubscriptionManager.Logoff
    Unload Me
    End
End Sub
```

```
' ------------------------------ ** ---------------------------------
'                               Private Functions
' ------------------------------ ** ---------------------------------

Private Sub Initialize()
    ' Create the event sink
    Set m_OneWorldTransientEventSink = New OneWorldTransientEventSink
End Sub

Private Sub EnableBizTalkIntegrationGroup()
    'Dim blnEnable As Boolean
    'blnEnable = (chkEnableBizTalkIntegration.Value = Checked)
    'lblScheduleFile.Enabled = blnEnable
    'txtScheduleFile.Enabled = blnEnable
End Sub
```

Integrating with BizTalk

This code is for the BizTalk integration for the received event.

Subscriber: BizTalk.cls

This code sample shows BizTalk subscription:

```
VERSION 1.0 CLASS
BEGIN
  MultiUse = -1  'True
  Persistable = 0  'NotPersistable
  DataBindingBehavior = 0  'vbNone
  DataSourceBehavior  = 0  'vbNone
  MTSTransactionMode  = 0  'NotAnMTSObject
END
Attribute VB_Name = "BizTalk"
Attribute VB_GlobalNameSpace = False
Attribute VB_Creatable = True
Attribute VB_PredeclaredId = False
Attribute VB_Exposed = False
Option Explicit

'*********************************************************
'***** ExecuteTutorial
'*****
'***** Purpose:  This component is used to exercise
'*****           the XLANG schedule portion of tutorial accompanying
'*****           BizTalk Server (this is the Module 1 Tutorial).
'*****           The component launches the specified schedule
'*****           file and passes the data file specified
'*****           to it using MSMQ.
'*****
'*****           NOTE: the source code in this component is a direct
```



```
'*****           adoption of the code found in the Module 1
'*****           Tutorial in the BizTalk Server 2000 documentation.
'*****           The default location for the original version of this
'*****           source is found in: C:\Program Files\Microsoft
'*****           BizTalk Server\Tutorial\Schedule\Solution\
'*****           ExecuteTutorial.vbp
'*****
'***** Inputs:
'*****          Schedule File - Contains the Moniker used to
'*****                          launch the schedule
'*****          Data File - Contains the location of the
'*****                      XML document to be passed to
'*****                      the schedule for processing.
'*****
'***** Outputs:
'*****          Data File - Data file is passed to MSMQ
'*****                      for later retrieval by the schedule.

Private g_MSMTxDisp As MSMQ.MSMQTransactionDispenser
Private g_MSMQQueue As MSMQ.MSMQQueue
Private g_MSMQInfo As MSMQ.MSMQQueueInfo
Private g_CurSkedDir As String
Private g_CurDataDir As String

Private Sub Class_Initialize()
    Set g_MSMQInfo = CreateObject("MSMQ.MSMQQueueInfo")
    Set g_MSMTxDisp = CreateObject("MSMQ.MSMQTransactionDispenser")
End Sub

Public Sub RunSchedule(ByVal strScheduleFile As String, ByVal
strData As String)
  Dim objfs As New FileSystemObject
  On Error GoTo cmdRunSked_Click_err

  'Connect To MSMQ and Remove Any Existing Messages
  PurgeMSMQ "DIRECT=OS:.\private$\ReceivePoReq"

 'Send Selected message to MSMQ
  ExecuteMSMQ "DIRECT=OS:.\private$\ReceivePoReq", strData

  'Start Schedule which reads message from MSMQ
  ExecuteSchedule strScheduleFile

  Exit Sub

cmdRunSked_Click_err:
    MsgBox Err.Description & vbCrLf & "Error: " & Err.Number & "
(0x" & Hex(Err.Number) & ")", vbCritical, "Error " & Err.Source
    Err.Clear
```

```
End Sub

Private Sub PurgeMSMQ(ByVal strQueuePath As String)
    Dim l_MSMQMsg As MSMQMessage

    On Error GoTo Err_ConnectMSMQ
    g_MSMQInfo.FormatName = strQueuePath
    Set g_MSMQQueue = g_MSMQInfo.Open(MQ_RECEIVE_ACCESS, MQ_DENY_NONE)

    On Error GoTo Err_PurgeMSMQ
    Do
        Set l_MSMQMsg = g_MSMQQueue.Receive(, , , 1)
    Loop While Not l_MSMQMsg Is Nothing
    Exit Sub

Err_ConnectMSMQ:
    Err.Raise Err.Number, "Connecting To MSMQ", "Could Not Open the
MSMQ Queue """ & strQueuePath & """." & vbCrLf & vbCrLf &
Err.Description

    Exit Sub
Err_PurgeMSMQ:
    Err.Raise Err.Number, "Cleaning MSMQ", "Could Not Remove
Existing Messages from MSMQ Queue """ & strQueuePath & """." &
vbCrLf & vbCrLf & Err.Description
    Exit Sub
End Sub

Private Sub ExecuteMSMQ(ByVal strQueuePath As String, DataToQueue
As  String)
    Dim QueueMsg As New MSMQMessage

    Dim strData As String
    Dim fSend As Boolean
    Dim txt As TextStream
    Dim mybyte() As Byte

    On Error GoTo Err_SendMSMQ
    g_MSMQInfo.FormatName = strQueuePath
    Set g_MSMQQueue = g_MSMQInfo.Open(MQ_SEND_ACCESS, MQ_DENY_NONE)
    mybyte = StrConv(DataToQueue, vbFromUnicode)
    QueueMsg.Body = DataToQueue

    Dim MSMQTx As Object
    Set MSMQTx = g_MSMTxDisp.BeginTransaction
    QueueMsg.Send g_MSMQQueue, MSMQTx
    MSMQTx.Commit

    Set QueueMsg = Nothing
    Set MSMQTx = Nothing
```



```
    Exit Sub

Err_SendMSMQ:
    Err.Raise Err.Number, "Sending Message To MSMQ", "Could Not
Send Message To MSMQ Queue """ & strQueuePath & """." & vbCrLf &
vbCrLf & Err.Description
    Exit Sub
End Sub

Private Sub ExecuteSchedule(ByVal strSchedule)
    Dim SendPAQ As Object
    On Error GoTo Err_ExecSched

    Set SendPAQ = GetObject(strSchedule)
    If SendPAQ Is Nothing Then
        Err.Raise vbObjectError + 1, , "Invalid Schedule Handle
Returned."
    End If
    Set SendPAQ = Nothing
    Exit Sub

Err_ExecSched:
    Err.Raise Err.Number, "Starting Schedule", "Could Not Launch
the XLANG Schedule" & vbCrLf & "Please verify the path to the SKX
file and the path to the data are correct. Also make sure the private
queues have been created." & vbCrLf & vbCrLf & Err.Description
    Exit Sub
End Sub
```

Adding a New Application

From the Microsoft Windows 2000 machine, navigate to COM+ Applications (Control Panel > Administrative Tools > Component Services), and then expand these buttons and folders:

Component Services > Computers > My Computer > COM+ Applications

To add a new application:

1. On Component Services, select COM+ Applications.
2. Right-click COM+ Applications, select New, and then select Application.

   The COM Application Install Wizard appears. These steps apply to the wizard.
3. On Install or Create a New Application, select Create an empty application.
4. On Create Empty Application, enter the name of the application (for example, JDECOMConnectorEvents).
5. Select an option for Activation Type, and then click Next.
6. On Set Application Identity, select the Interactive User option, and then click Next.
7. Click Finish.

A new application, with the name you entered in Step 4, is added to COM+ Applications.

Installing the Event Class

On Component Services, expand the folder for the new application (for example, JDECOMConnectorEvents).

To install the event class:

1. On Component Services, select Components.
2. Right-click Components, select New, and then select Component.

   The COM Component Install Wizard appears. These steps apply to the wizard.
3. On Import or Install a Component, select Install new event class(es).
4. On Select Files to Install, browse to the EventClass.dll on the Microsoft Windows 2000 machine.
5. Select EventClass.dll, and then click Open.

   Install new event class appears with information in these fields:
   - Files to install
   - Event classes found
6. Click Next, and then click Finish.

EventClass.dll is successfully added to Component Services.

Registering EventSink for Persistent Subscription

After you register an event class in the COM+ catalog, you can add subscribers to the event class and subscriptions to the subscribers. For persistent event subscription:

- Add a new application for EventSink.
- Install the type library component for EventSink.
- Add a subscription.

Note. To add EventSink, follow the steps in the task named To add a new application in the Connectors Guide. The name of the application is EventSink, or a name that you prefer.

To install the EventSink component:

On Component Services, expand the folder for the new application (for example, EventSink).

1. Select Components.
2. Right-click Components, select New, and then select Component.

   The COM Component Install Wizard appears. These steps are for the wizard.
3. On Import or Install a Component, select Install new component(s).
4. On Select Files to Install, browse to the EventSink.dll that you previously developed.
5. Select EventSink.dll, and then click Open.

   Install new component appears with information in these fields:
   - Files to install

- Event classes found

6. Click Next, and then click Finish.

EventSink.dll is successfully added to Component Services.

To add a subscription:

In COM+ Applications, expand these folders:

JDECOMConnectorEvents > Components > EventSink.OneWorldTransientEventSink

1. Select Subscription.
2. Right-click Subscription, select New, and then select Subscription.

   The COM New Subscription Wizard appears. These steps apply to the wizard.
3. On Select Subscription Method(s), chose IOWEvent, and then click Next.
4. If appropriate, select the Use all interfaces for this component option.
5. On Select Event Class, select the event class (for example, JDEdwards.EventClass.OneWorldEventClass.1), and then press Next.

   If multiple EventSink classes have implemented the event interface, then use all event classes that implement that specified interface. If only one EventSink class has implemented the event interface, then just select that specific class.
6. On Subscription Options, enter the name of the subscription (for example, MySubscription).
7. In the Options area, select the Enable this subscription immediately option, and then click Next.
8. Click Finish.

   A new subscription, with the name you entered in Step 6, is added to COM+ Services. You must define the name of the event for the subscription.
9. Right-click the subscription (for example, MySubscription), and then select Properties.
10. On MySubscription Properties, click the Options tab.
11. Chose the Enabled option.
12. In the Filter criteria field, enter the name of the event for which you want a subscription.

    Enter all of the events for which you want to subscribe. The filter criteria string supports relational operations (=, ==, !, !=, ~, ~=, <>), nested parentheses, and logical words (AND, OR, and NOT); for example:

    EventName==’RTSOOUT’ OR EventName==RTPOOUT’
13. Click OK.

CHAPTER 7

Understanding jdeinterop.ini for COM Connector

This chapter discusses the jdeinterop.ini file for the COM connector.

Settings for jdeinterop.ini File for the COM Connector

The jdeinterop.ini file includes settings the server might need. The default location for the file is `c:\`; however, you can configure this location. Information is organized by section, for example [JDENET].

These sections are configured for the COM connector:

- OCM
- JDENET
- Server
- Security
- Debug
- Interop
- Events
- JMSEVENTS (Only for guaranteed events delivery method using JD Edwards EnterpriseOne Tools 8.94)

Note. Unless otherwise indicated, the sections and the settings for the COM connector are for all JD Edwards EnterpriseOne releases.

[OCM]

Configure these [OCM] settings for the COM connector:

Setting and Typical Value	Purpose	Applicable Release
DSN=ODA ITTND17	The data source name from the system DSN of the ODBC setting.	All
OCM Datasource=COM OCM	System data source for JD Edwards EnterpriseOne client.	All
DB User=jde	User for the data source connection.	All
DB Pwd=jde	Password for the data source connection.	All

Setting and Typical Value	Purpose	Applicable Release
Object Owner=sysb9	For UNIX platforms, this is the object owner in the [DB SYSTEM SETTINGS].	All
Seperator=.	Separator used in SQL query. For Oracle, SQL, and UDB databases, the separator is period (.); for iSeries, the separator is a slash (/).	All

[JDENET]

Configure these [JDENET] settings for the COM connector:

Setting and Typical Value	Purpose	Applicable Release
enterpriseServerTimeout= 90000	Timeout value for a request to the JD Edwards EnterpriseOne enterprise server.	All
maxPoolSize=30	JDENET socket connection pool size.	All

[SERVER]

Configure these [SERVER] settings for the COM connector:

Setting and Typical Value	Purpose	Applicable Release
glossaryTextServer=JDED: 6010	The JD Edwards EnterpriseOne enterprise server and port that provide glossary text information.	All
codePage=1252	The encoding scheme, such as: 1252 English and Western European. 932 Japanese. 950 Traditional Chinese. 936 Simplified Chinese. 949 Korean.	All

[SECURITY]

Configure this [SECURITY] setting for the COM connector.

Setting and Typical Value	Purpose	Applicable Release
NumServers=1	Number of security servers set.	All

[DEBUG]

Configure these [DEBUG] settings for the COM connector:

Setting and Typical Value	Purpose	Applicable Release
JobFile=c:\Interop.log	Location of error file.	All
DebugFile=c:\InteropDebug.log	Location of debug file.	All
log=c:\net.log	Location of log file.	All
debugLevel=0 - 12	Defines the level of tracing provided by the COM connector and the CallObject component in the specified log file, in the COM server only. 0 None: Logging is turned off and only errors are written to the JobFile. 2 Errors (error messages). 4 System Errors (exception messages). 6 Warning Information. 8 Min Trace (Key operations; for example, Login, Logoff, Business Function calls). 10 Trouble Shooting Information (Help). 12 Complete Debug Information (Logs everything). **Note.** The odd values are reserved for future levels to be added. You typically do not need to use tracing. However, tracing is useful for debugging.	All
netTraceLevel=0	Defines the level of tracing provided by the ThinNet component in the specified log file, in the COM server only. 0 No trace. 1 Record process ID, thread ID, and the available socket status when a new connection is added and the socket pool is searched. 2 Includes the information in trace level 1 and also traces every call made in the Connection Manager class. 3 Includes all information in trace level 2, and also traces getPort calls and getHost calls. **Note.** You typically do not need to use tracing. However, tracing is useful for debugging.	All

[INTEROP]

Configure these [INTEROP] settings for the COM connector:

Setting and Typical Value	Purpose	Applicable Release
SettingTime=10	Enables the connector to access and retrieve event information from the F90703 and F90704 tables. Defines the time for the connector applications to start up before the connector starts recovering an event. This value is seconds.	JD Edwards EnterpriseOne Tools 8.93 with ESU
RecoveryInterval=60	Enables the connector to access and retrieve event information from the F90703 and F90704 tables. Defines the time for the connector applications to start up before the connector starts recovering an event. This value is seconds.	JD Edwards EnterpriseOne Tools 8.93 with ESU
enterpriseServer=JDED	The JD Edwards EnterpriseOne server.	All
port=6010	The port number of the JD Edwards EnterpriseOne server.	All
manual_timout=300000	The time-out value for a transaction in manual commit mode.	All
Repository=c:\JDEdwards\Interop\repository	Points to the location of the repository directory containing business object libraries (generated JAR files).	All

[EVENTS]

Configure these [EVENTS] settings for the COM connector:

Setting and Typical Value	Purpose	Applicable Release
UseGuaranteedEvents System= True	Indicates guaranteed event delivery. Values are true and false. Must be set to true when using JD Edwards EnterpriseOne Tools 8.95 and later Tools releases with JD Edwards EnterpriseOne Applications 8.9 and later Applications releases, and you want to use guaranteed event delivery.	JD Edwards EnterpriseOne Tools 8.95 and later Tools releases with JD Edwards EnterpriseOne Applications 8.9 and later Applications releases.
Transport=HTTP	Defines the event transport mechanism. Valued values are HTTP and JMS. the default value is HTTP.	JD Edwards EnterpriseOne Tools 8.95 and later Tools releases with JD Edwards EnterpriseOne Applications 8.9 and later Applications releases.

Setting and Typical Value	Purpose	Applicable Release
eventServiceURL=http://hpdev1:9081/e1events/EventClientService	Locates the event service. If the value for the Transport= setting is HTTP, then this setting must be configured.	JD Edwards EnterpriseOne Tools 8.95 and later Tools releases with JD Edwards EnterpriseOne Applications 8.9 and later Applications releases.
jndiProviderURL= jndiProvider URL= corbaloc::denmlps14.mlab.jdedwards.com:9810/NameServiceServerRoot	Locates the event service. If the value for the Transport= setting is JMS, then this setting must be configured.	JD Edwards EnterpriseOne Tools 8.95 and later Tools releases with JD Edwards EnterpriseOne Applications 8.9 and later Applications releases.
eventReceiveTimeout=60000	Maximum number of milliseconds that the event receiver waits before unsubscribing the event from the JD Edwards EnterpriseOne server.	JD Edwards EnterpriseOne Tools 8.95 and later Tools releases with JD Edwards EnterpriseOne Applications 8.9 and later Applications releases.
initialContextFactory= com.ibm. websphere.naming.WsnInitialContextFactory	The initial Context Factory	JD Edwards EnterpriseOne Tools 8.94 with JD Edwards EnterpriseOne Applications 8.11 and COM connection is through WebSphere.
jndiProviderURL=corbaloc::<server_name:server_port/NameServiceServerRoot	Replace <server_name:server_port> with actual values relevant to the WebSphere server. A common value for the server_port for WebSphere is 9810, but consult the WebSphere administrator to confirm this port value.	JD Edwards EnterpriseOne Tools 8.94 with JD Edwards EnterpriseOne Applications 8.11 and COM connection is through WebSphere.
port=6002	The socket port number where the EventListener receives the events from the JD Edwards EnterpriseOne server. This port should not be used by any other resource. Also, the port should not be changed dynamically when the connector is running, as this causes subsequent subscriptions to be lost.	All

Setting and Typical Value	Purpose	Applicable Release
ListenerMaxConnection=10	The maximum number of connections allowed by the EventListener. The default number of connections is 10, but you can change this number. The maximum number of connections allowed is 64.	All
ListenerMaxQueueEntry=10	The maximum number of events that the EventListener can hold before processing by the EventManager. The default number of events for the queue is 10, but you can change this number. The maximum number of events that can be held in the queue is 100.	All
Outbound_timeout=1200000	Maximum number of milliseconds that the EventManager waits before unsubscribing the transient event from the JD Edwards EnterpriseOne server.	All

[JMSEVENTS]

Use this section only if you use the COM connector with JD Edwards EnterpriseOne Tools 8.94 and JD Edwards EnterpriseOne Applications 8.11.

This section has a single setting, CLASSPATH. Note that you must include the full directory path of each file, separating each file by a semicolon. For example, CLASSPATH=connector.jar;log4j.jar;System_JAR.jar.

These files can be found in the <JD Edwards EnterpriseOne Windows client installation directory>\system\classes folder:

- connector.jar
- log4j.jar
- base_JAR.jar
- jdeNet_JAR.jar
- System_JAR.jar
- jdbjBase_JAR.jar
- jdbjInterfaces_JAR.jar

These files can be found on the <Transaction server installation directory>\EventProcessor\app folder:

- EventProcessor_EJB.jar
- EventProcessor_JAR.jar

Note. The files on the client side and Transaction server side must always match. This is important if the Transaction server is updated.

- The path to the directory where the jdeinterop.ini, jdbj.ini, and jdelog.properties files exist, which must all be in one directory.

  This CLASSPATH entry must end with a slash (\), which indicates it is a directory name and not a file name.
- The full path to the JDBC driver files, including the filenames.

WebSphere

If you use WebSphere for the Java connection, you must include additional files. Note that IBM WebSphere MQ is normally included as part of other WebSphere applications, including the WebSphere Application Client.

These files are normally located in the <IBM WebSphere MQ installation directory>/Java/lib folder:

- com.ibm.mqjms.jar
- com.ibm.mq.jar
- com.ibm.mqbind.jar

These files are normally in the <WebSphere installation directory>\lib folder:

- bootstrap.jar
- j2ee.jar
- Improxy.jar
- urlprotocols.jar
- ecutils.jar
- messagingClient.jar
- naming.jar
- namingclient.jar

You must also include the <WebSphere installation directory>/properties directory in the CLASSPATH.

CHAPTER 8

Understanding Java Interoperability Solution

This chapter provides an overview of the Java interoperability solution.

Java Interoperability Solution

The JD Edwards EnterpriseOne Java interoperability solution enables you to write Java applications that interact with the JD Edwards EnterpriseOne system. The Java interoperability solution includes these types of connectors:

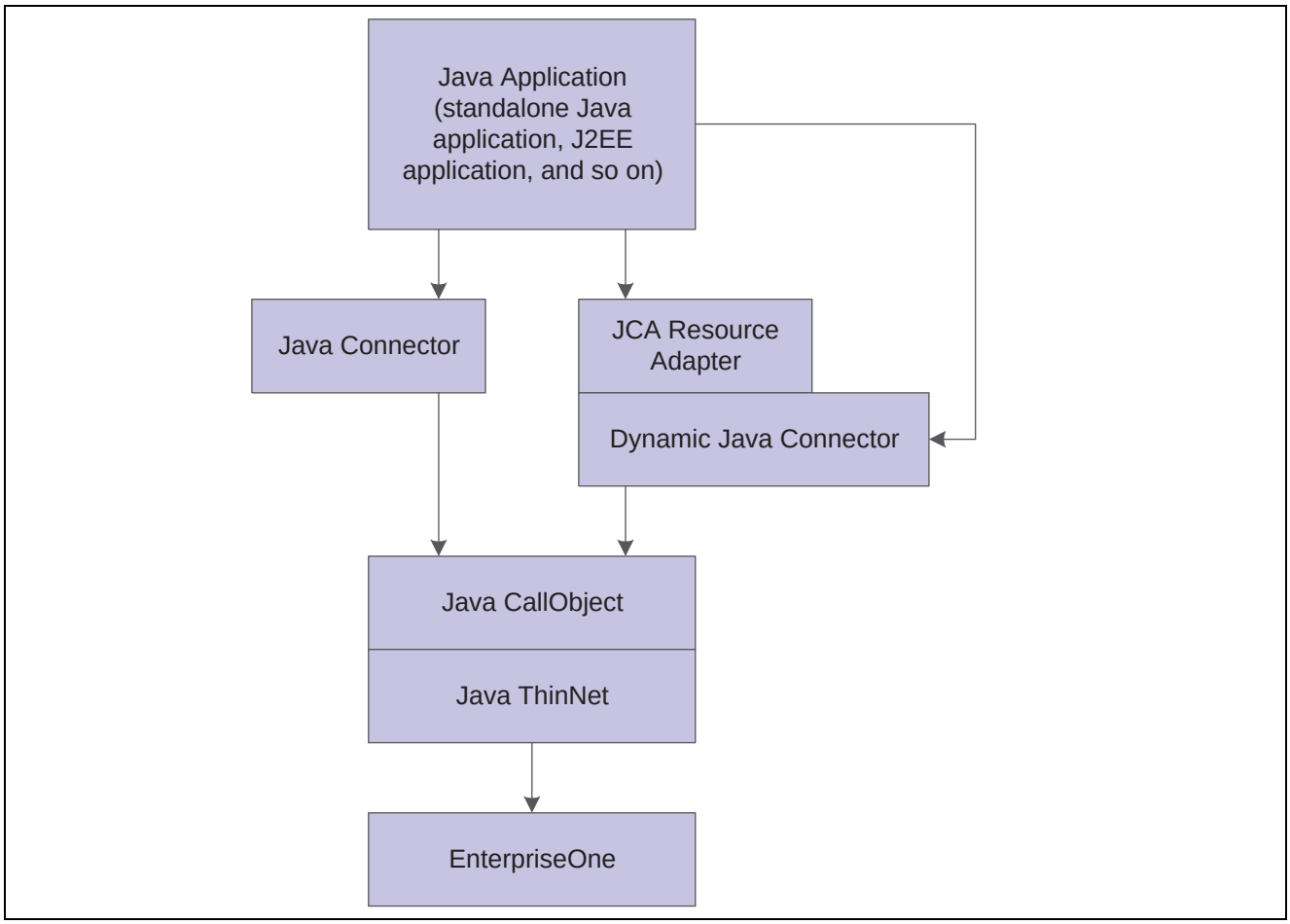
- Dynamic Java connector.
- Java connector.
- Java Connector Architecture (JCA) resource adapter.

The initial Java interoperability solution provided is the Java connector. The Java connector generates a Java wrapper object around the JD Edwards EnterpriseOne business function and data structure. A Java application calls the business functions from the Java wrapper object.

The dynamic Java connector is an enhancement to the Java connector. The dynamic Java connector enables Java applications to dynamically call business functions without generating business function wrappers. The dynamic Java connector ensures that the Java business function is compatible with the server spec. The dynamic Java connector makes it much easier for the Java application to switch between JD Edwards EnterpriseOne environments.

The JCA resource adapter is a thin layer built on top of the dynamic Java connector and provides standard APIs required by the Java connector architecture. The core functionality for the JCA resource adapter is to interact with JD Edwards EnterpriseOne, and this functionality is leveraged to the dynamic Java connector. Each connector has a complete set of APIs that enable Java applications to interact with JD Edwards EnterpriseOne.

This diagram shows how a Java application interacts with JD Edwards EnterpriseOne through a connector:



Java Application interaction with JD Edwards EnterpriseOne

Generally, each connector provides public interfaces (or APIs) for these services that can be used by a Java application:

Service	Description
Security Management	Handles security access to the JD Edwards EnterpriseOne system.
User Session Management	Manages the user session pooling.
Business Function Calls	How the Java application calls business functions.
Transaction Management	Manages the transaction process to the JD Edwards EnterpriseOne system.
Error Handling	Provides the appropriate exceptions to the connector user to easily handle error scenarios.

Both the Java connector and the dynamic Java connector support the processing of outbound events.

Note. If this is the first implementation of a Java connector, you should consider the dynamic Java connector instead of the Java connector. The functional capabilities are the same. The advantage of implementing the dynamic Java connector is that you are not required to generate wrappers.

CHAPTER 9

Working with the Dynamic Java Connector

This chapter provides an overview of the dynamic Java connector and discusses how to:

- Design the dynamic Java connector.
- Install the dynamic Java connector.
- Run the dynamic Java connector.
- Manage the user session for the dynamic Java connector.
- Use Sample applications.

Understanding the Dynamic Java Connector

The dynamic Java connector enables a Java application to call a business function. Compared to the Java connector, the dynamic Java connector has these distinguishing features:

- Dynamically introspects business function metadata.

  The business function metadata is introspected from the JD Edwards EnterpriseOne server during application design time by using connector APIs without pre-generating business function wrappers.

- Dynamically calls business functions without pre-generating business function wrappers.

  Since there is no local storage of business function spec metadata, the business function used by the dynamic Java connector is always compatible with the server spec metadata.

- Easily switches from one environment to another environment.

  The Java application can run on any environment that is compatible to the environment on which the Java application was designed.

The dynamic Java connector provides these services:

- For application design, the dynamic Java connector permits client programs to introspect business function specification metadata.
- For application deployment, the dynamic Java connector validates whether a client application can run through a certain JD Edwards EnterpriseOne server.
- For application runtime, the dynamic Java connector provides an interface that permits the connector client to call the business function on the JD Edwards EnterpriseOne server.

Each server is described in detail in corresponding sections of this guide.

Designing the Dynamic Java Connector

This section provides considerations for designing the dynamic Java connector and discusses:

- Business function spec metadata introspection.
- Business function spec metadata validation.
- SpecImage console.

Business Function Spec Metadata Introspection

To call a business function method, you need to know the business function methods that are available to be called, and you need to know about the business function metadata. This list provides examples of metadata:

- Business function method (such as F4211BeginDoc).
- The module name (C file name) to which a business function method belongs (such as B123456).
- Description of the business function method (such as sales order).
- Data structure template name that is associated with a business function method (such as D123456).
- The attributes for all of the data items (parameters) in a business function method, such as name=szMnAddressbookNumber, itemID=1, data type=Math_Numeric, length=48, requiredType="Yes", IOType="INOUT".

In the dynamic Java connector, metadata is represented by the BSFNMethod and BSFNParameter interfaces.

BSFNMethod

The BSFNMethod interface defines APIs that enable you to retrieve metadata related to the business function method. The BSFNMethod interface defines these APIs:

- public String getName();
- public String getDSTemplateName();
- public String getBSFNName();
- public String getDescription();
- public BSFNParameter getParameter(String paraName);
- public BSFNParameter[] getParameters();
- public String getFormatString();
- public ExecutableMethod createExecutable();
- public boolean equals(Object anotherBSFNMethod);
- public void setEqualTo(BSFNMethod anotherBSFNMethod);
- public String getVersion();
- public void setVersion(String version);

BSFNParameter

The BSFNParameter interface defines APIs that enable you to retrieve metadata related to the data structure of the business function. The BSFNParameter interface defines these APIs:

- public int getItemID();
- public String getName();
- public int getLength();
- public IOType getIOType();
- public RequiredType getRequiredType();
- public BSFNDataType get DataType();

BSFNSpecSource

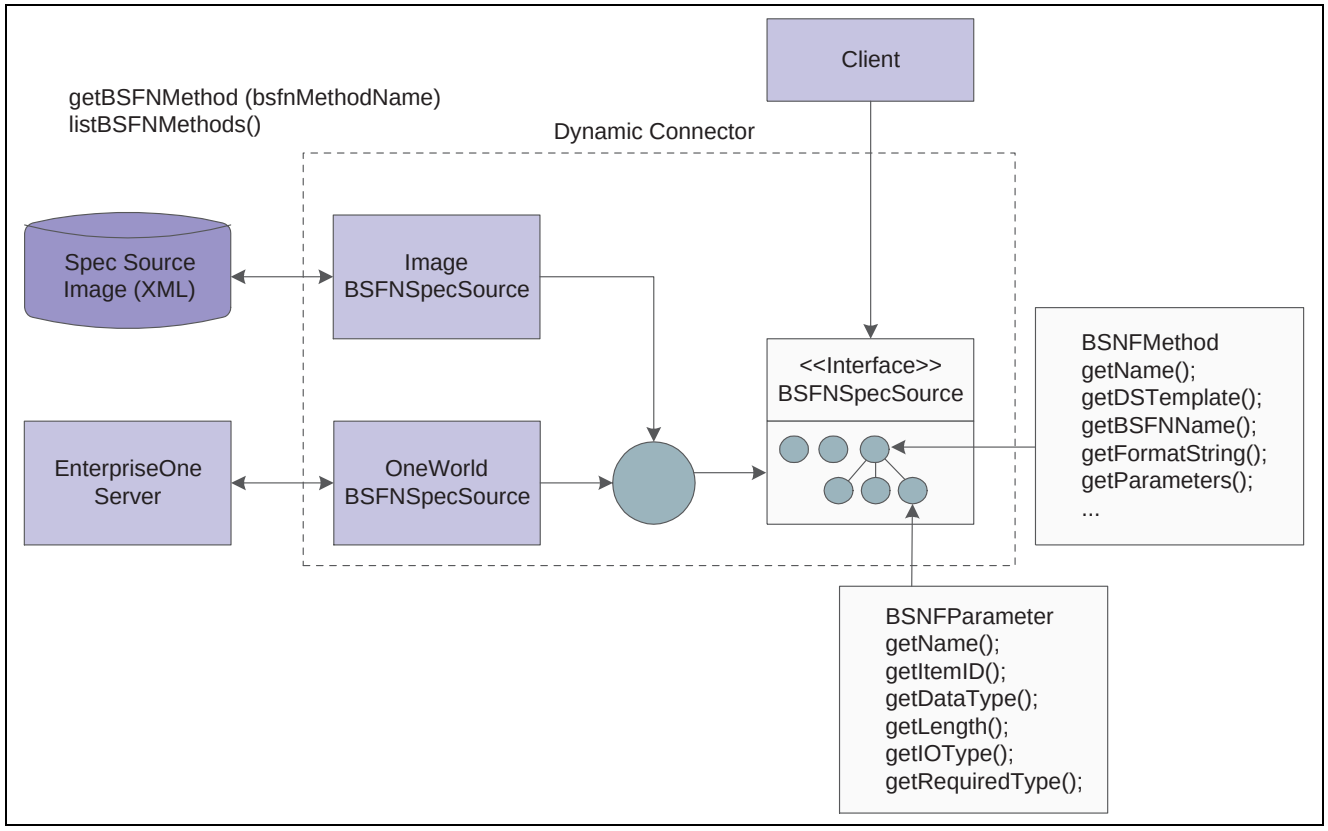
You can write a program to retrieve business function method metadata through an interface called BSFNSpecSource. The BSFNSpecSource interface defines these APIs:

- Public BSFNMethod getBSFNMethod(String methodName) throws SpecFailureException
- Public BSFNMethod[ ] getBSFNMethods() throws SpecFailureException

The class that implements the BSFNSpecSource interface reads the business function method metadata from an external physical repository and creates the BSFNMethod object. AbstractBSFNSpecSource is an abstract implementation of BSFNSpecSource provided by the dynamic Java connector. All customized implementations of BSFNSpecSource should be a subclass of this class. OneWorldBSFNSpecSource is the default implementation of AbstractBSFNSpecSource.

See Chapter 9, "Working with the Dynamic Java Connector," Installing the Dynamic Java Connector, page 97.

This illustration shows the BSFNSpecSource, BSFNMethod, and BSFNParameter relationships:



Relationships among BSFNSpecSource, BSFNMethod, and BSFNParameter

This code example shows how to retrieve the BSFN spec from BSFNSpecSource:

```
import com.jdedwards.system.connector.dynamic.spec.source.BSFNSpecSource;
import com.jdedwards.system.connector.dynamic.spec.source.OneworldBSFNSpecSource;
import com.jdedwards.system.connector.dynamic.Connector;
import com.jdedwards.system.connector.dynamic.spec.source.*;
import com.jdedwards.system.connector.dynamic.spec.SpecFailureException;
import com.jdedwards.system.connector.dynamic.ServerFailureException;

... //Declare class
 }
 public void execMethod() throws SpecFailureException,ServerFailureException
 {
   BSFNSpecSource specSource = null;
   int sessionID = Connector.getInstance().login("user", "pwd", "env","role");
   //specSource = new OneWorldBSFNSpecSource(sessionID); Problem in this
line. World should be small
   specSource = new OneworldBSFNSpecSource(sessionID);
   // or specSource = new ImageBSFNSpecSource("SSI.xml");
   //Step 2: Get BSFNMethod by name from specSource
   BSFNMethod method = specSource.getBSFNMethod("GetEffectiveAddress");
   String methodName = method.getName();
   System.out.println("Method name is "+methodName);
   BSFNParameter[] paraList = method.getParameters();

   for (int i=0; i<paraList.length;i++)
   {
    BSFNParameter para = paraList[i];
    String name=para.getName();
    System.out.println("Name is "+name);
   }
 }
```

SpecDictionary

A BSFNSpecSource can contain thousands of business function methods. The dynamic Java connector provides an interface to properly categorize and organize business function methods. Without proper categorization and organization, it is difficult to navigate and find the proper business function method. To solve this problem, the dynamic Java connector provides an interface called SpecDictionary, which provides these services:

- Categorizes business function methods in a hierarchy.
- Masks the BSFNSpecSource and limits the number of business function methods a client can view.

The entry of SpecDictionary is called a context. A context is a set of name-to-object bindings. Every context has an associated naming convention. A context provides a lookup operation that returns the object. The dynamic Java connector provides these two concrete classes that implement the SpecDictionary:

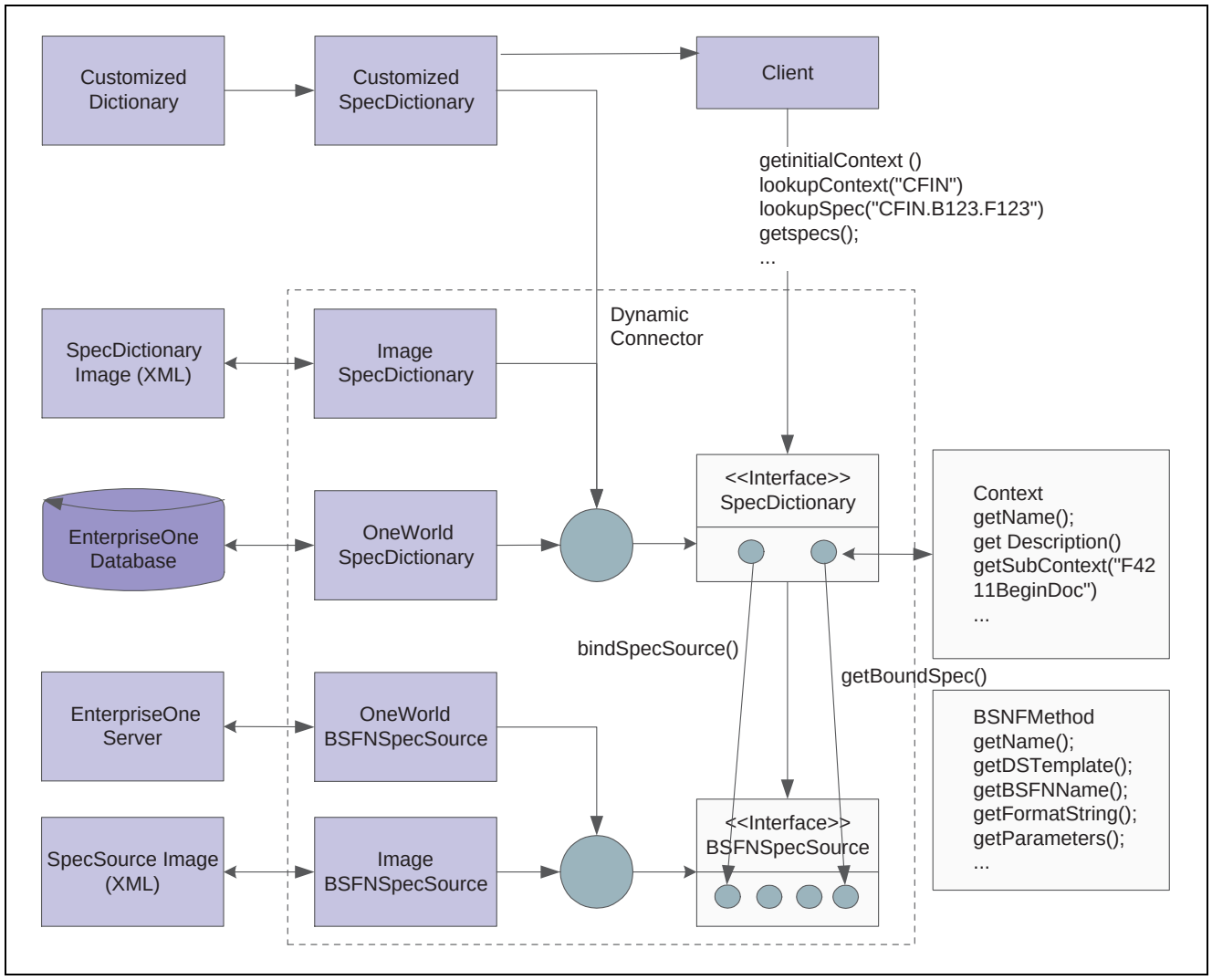
- OneWorldSpecDictionary, which gets the hierarchy information from the JD Edwards EnterpriseOne database.

  OneWorldSpecDictionary categorizes business function methods as DLL library - C file name - C function name.

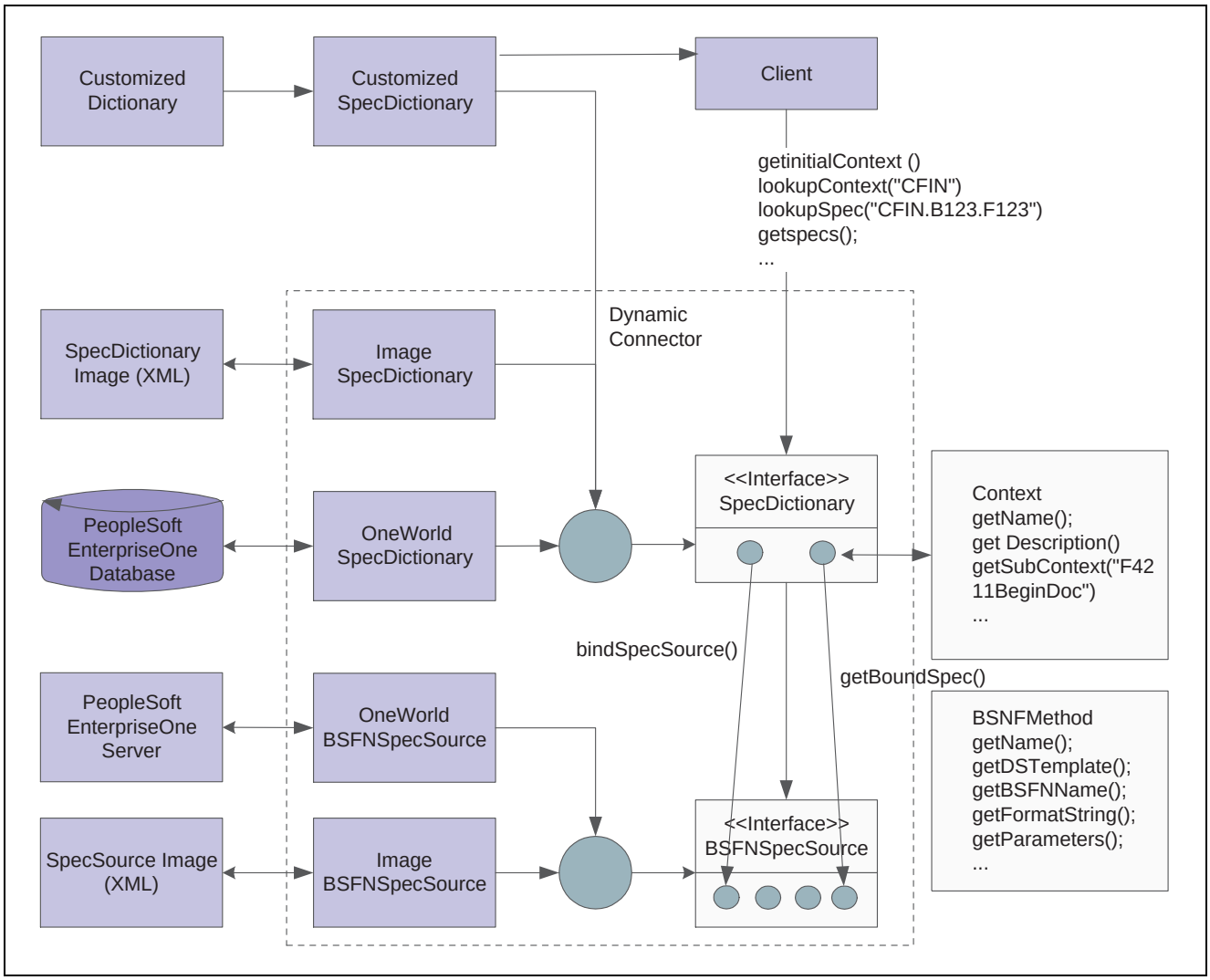
- ImagespecDictionary, which gets the hierarchy information from Spec Dictionary Image, which is an XML file.

Like BSFNSpecSource, third-party programs can store the spec dictionary information in their proprietary format, but they need to implement their own specDictionary to read the proprietary spec.

This diagram shows the relationship between SpecDictionary and BSFNSpecSource:



Relationship between SpecDictionary and BSFNSpecSource



Relationship between SpecDictionary and BSFNSpecSource

This example code shows how to use SpecDictionary and BSFNSpecSource to browse and lookup information:

```
import com.jdedwards.system.connector.dynamic.spec.source.BSFNSpecSource;
import com.jdedwards.system.connector.dynamic.spec.source.OneworldBSFNSpecSource;
import com.jdedwards.system.connector.dynamic.Connector;
import com.jdedwards.system.connector.dynamic.spec.source.*;
import com.jdedwards.system.connector.dynamic.spec.SpecFailureException;
import com.jdedwards.system.connector.dynamic.ServerFailureException;
import com.jdedwards.system.connector.dynamic.spec.dictionary.Context;
//import com.jdedwards.system.connector.dynamic.spec.dictionary.
InvalidBindingException;
import com.jdedwards.system.connector.dynamic.spec.dictionary.SpecDictionary;
import com.jdedwards.system.connector.dynamic.spec.dictionary.
OneworldSpecDictionary;

... //Declare Class
 }
```

```
 public void execMethod() throws SpecFailureException,ServerFailureException
 {
  BSFNSpecSource specSource = null;
  SpecDictionary specDictionary = null;

  //Step 1: Create a SpecDictionary
  int sessionID = Connector.getInstance().login("user", "pwd", "env","role");
  specDictionary = new OneworldSpecDictionary(sessionID);
  // or specDictionary = new ImagespecDictionary("dict.xml");

  //Step 2: Bind the SpecDictionary to a SpecSource
  specDictionary.bindSpecSource(specSource);

  //Step 3a: Lookup the BSFNMethod by giving the full path
  //Problem in this line. Extra braces // BSFNMethod method =(BSFNMEthod)
specDictionary.getSpec("CFIN.F4211.F4211BeginDoc"));
//Class name is wrong  BSFNMethod method =(BSFNMethod) specDictionary.
getSpec("CFIN.F4211.F4211BeginDoc");
  BSFNMethod method =(BSFNMethod) specDictionary.getSpec("CFIN.F4211.
F4211BeginDoc");

  //Step 3b: or navigate through the dictionary and get the context attributes
  Context initContext = specDictionary.getInitialContext();
  Context[] subContextList = initContext.getSubcontexts();
  //Illegal expression // for (int I=0;I<subContextList>.length; I++)
  for (int I=0;I<subContextList.length; I++)
  {
   Context subContext=subContextList[I];
   subContext.getName();
   subContext.getDescription();
   method=(BSFNMethod)subContext.getBoundSpec();
  }
 }
```

Business Function Spec Metadata Validation

If the dynamic Java connector program calls a business function from OneWorldBSFNSpecSource, you do not need to validate the business function metadata. The business function metadata in OneWorldBSFNSpecSource is always the same as the business function metadata that is on the JD Edwards EnterpriseOne server where the business function runs. You must ensure that all input parameters are set correctly, according to OneWorldBSFNSpecSource.

If the dynamic Java connector program calls a business function from a spec source other than OneWorldBSFNSpecSource (such as ImageBSFNSpecSource or a custom business function spec source), the business function metadata that is in the local spec source might not be compatible with the business function metadata that is on the JD Edwards EnterpriseOne server where the business function runs. Local business function spec metadata can be validated during these conditions:

Condition	Explanation
Deploy Time	The dynamic Java connector program validates the local spec source against the JD Edwards EnterpriseOne server spec source before run time. You should perform this validation, as all business functions in the local spec source are validated. The program can be redesigned before it is shipped.
Run Time	The dynamic Java connector validates the program based on the local spec design when running business functions. During this condition, only the business function that is called is validated. Run time validations should be treated as error handling when incompatible business function specs are found.

The dynamic Java connector provides two ways to validate business function spec metadata during deploy time: SpecImageValidator APIs and SpecImageConsole command line.

The APIs for SpecImageValidator are:

- public SpecImageValidator(BSFNSpecSource srcSpecSource).
- public ValidationResultSet validate(SpecDictionary dictionary) throws SpecFailureException.
- public ValidationResultSet validate(SpecDictionary dictionary, String path) throws SpecFailureException.
- public ValidationResultSet validate(BSFNSpecSource dstSpecSource) throws SpecFailureException.
- public ValidationResultSet validate(BSFNSpecSource dstSpecSource, String bsfnMethodName).

Note. If the SpecImageConsole command line is used, the dynamic Java connector can only validate business function spec metadata from ImageBSFNSpecSource; custom business function spec sources cannot be validated.

SpecImageConsole

You can use the SpecImageConsole command line to generate, update, validate and synchronize spec images.

Generate Spec Image

You use the spec image console to generate or regenerate a spec image. This information is useful for generating or regenerating a spec image.

Usage

java com.jdedwards.system.connector.dynamic.util.SpecImageConsole /Generate [Other Options]

Options

/UserName <user> (required)

/Password <pwd> (required)

/Env <environment> (required)

/Role <role> (required)

/ImageStub <stub file> (required)

/ImageType <image type [SSI|SDI|ALL]> (optional, default is ALL)

/ErrorFile <error file> (optional, default is System.err)

/OutputFile <output file> (optional, default is System.out)

Explanation

Log on to JD Edwards EnterpriseOne with <user>, <pwd>, <environment>, and <role>.

Load the spec image stub from <stub file>.

Generate the spec image with the image type <image type>.

The spec image is written to the <output file> (or System.out if /OutputFile not present).

Error messages are written to the <error file> (or System.err if /ErrorFile not present).

Example

This shows example code:

```
java com.jdedwards.system.connector.dynamic.util.SpecImageConsole
/Generate /ImageStub image_stub.xml /ImageType SDI /OutputFile
image.xml /ErrorFile err.log
```

Update Spec Image

You use the spec image console to update or change a spec image. This information is useful for updating a spec image.

Usage

java com.jdedwards.system.connector.dynamic.util.SpecImageConsole /Update [Other Options]

Options

/UserName <user> (required)

/Password <pwd> (required)

/Env <environment> (required)

/Role <role> (required)

/SSI <SSI file> (required)

/SDI <SDI file> (optional)

/AddSpec <BSFNSpec name> (for example, F4211BeginDoc; optional)

/AddContext <full Context name> (for example, CFIN.B3100010 or CFIN.B3100010.F4211BeginDoc; optional)

/RemoveSpec <BSFNSpec name> (for example, F4211BeginDoc; optional)

/RemoveContext <full Context name> (for example, CFIN.B3100010 or CFIN.B3100010.F4211BeginDoc; optional)

Explanation

Log on to JD Edwards EnterpriseOne with <user>, <pwd>, <environment>, and <role>.

Load the <SDI file> (If option /SDI not present, then load <SSI file>) add/remove the context and BSFN spec that is specified as <full Context name> and <BSFNSpec name>.

Example

This example shows how to update the Spec Dictionary Image (sdi.xml) and the Spec Content Image (SSI.xml). The example adds Context CFIN.B00100, removes Context CFIN.B001002, adds Spec F4211BeginDoc, and removes Spec F4311BeginDoc.

```
java com.jdedwards.system.connector.dynamic.util.SpecImageConsole
/Update /SDI sdi.xml /SSI ssi.xml /addContext CFIN.B001001
/removeContext CFIN.B001002 /addSpec F4211BeginDoc /removeSpec
F4311BeginDoc
```

Validate Spec Image

You use the spec image console to validate the spec image against the JD Edwards EnterpriseOne server. This information is useful for validating a spec image.

Usage

java com.jdedwards.system.connector.dynamic.util.SpecImageConsole /Validate [Other Options]

Options

/UserName <user> (required)

/Password <pwd> (required)

/Env <environment> (required)

/Role <role> (required)

/SSI <SSI file> (required)

/SDI <SDI file> (optional)

/OutputFile (optional, default to System.out)

Explanation

Log on to JD Edwards EnterpriseOne with <user>, <pwd>, <environment>, and <role>.

If option /SDI is present, validate all the BSFNSpec that bind to the <SDI file>. If /SDI is not present, validate all the BSFNSpec in the <SSI file>.

The spec image is written to the <output file> (or System.out if /OutputFile is not present).

Example

This example shows how to validate spec image using ssi.xml as the SpecDictionary and sdi.xml as the SpecSource. The example writes the validation result to validateResult.log.

```
java com.jdedwards.system.connector.dynamic.util.SpecImageConsole
/Validate /SDI sdi.xml /SSI ssi.xml /OutputFile validateResult.log
```

Synchronize Spec Image

You use the spec image console to synchronize the spec image with the JD Edwards EnterpriseOne server. This information is useful for validating a spec image.

Usage

java com.jdedwards.system.connector.dynamic.util.SpecImageConsole /Synchronize [Other Options]

Options

/UserName <user> (required)

/Password <pwd> (required)

/Env <environment> (required)

/Role <role> (required)

/SSI <SSI file> (required)

/SDI <SDI file> (optional)

/ErrorFile <err file>(optional, default to System.err)

Explanation

Log on to JD Edwards EnterpriseOne with <user>, <pwd>, <environment>, and <role>.

If option /SDI present, synchronize all the BSFNSpec that bind to the <SDI file>. If /SDI is not present, synchronize all the BSFNSpec in the <SSI file>.

The new spec image is written to the <SSI file>. Error messages are written to <err file> (or System.err if /ErrorFile is not present).

Example

This example shows how to synchronize the spec source image, ssi.xml:

```
java com.jdedwards.system.connector.dynamic.util.SpecImageConsole
/Synchronize /SSI ssi.xml
```

Installing the Dynamic Java Connector

These steps illustrate how to install dynamic connector components so that you can run a dynamic Java connector application.

1. Copy these files from the JD Edwards EnterpriseOne server to a directory on the machine that you want to use:
   - connector.jar
   - jdbjBase_JAR.jar
   - jdbjInterfaces_JAR.jar
   - jdeNet_JAR.jar
   - base_JAR.jar
   - spec_JAR.jar
   - System_JAR.jar

- log4j.jar
- xalan.jar
- xerces.jar
- PMApi_JAR.jar
- BizLogicContainer_JAR.jar
- BizLogicContainerClient_JAR.jar
- ApplicationAPIs_JAR.jar
- ApplicationLogic_JAR.jar
- jdeinterop.ini
- jdbj.ini
- jdelog.properties
- JDBC drivers (obtain the JDBC drivers from the database vendor)

  For example, you might copy the files to this directory on the machine:

  C:\JDEdwards\Interop

2. Add these files to the CLASSPATH:
   - connector.jar
   - jdbjBase_JAR.jar
   - jdbjInterfaces_JAR.jar
   - jdeNet_JAR.jar
   - base_JAR.jar
   - Spec_JAR.jar
   - System_JAR.jar
   - log4j.jar
   - xalan.jar
   - xerces.jar
   - PMApi_JAR.jar
   - BizLogicContainer_JAR.jar
   - BizLogicContainerClient_JAR.jar
   - ApplicationAPIs_JAR.jar
   - ApplicationLogic_JAR.jar
   - JDBC drivers
3. Add the path where the jdelog.properties, jdeinterop.ini, and jdbj.ini files are located into CLASSPATH.
4. Edit jdeinterop.ini, jdelog.properties, and jdbj.ini for proper settings.

Note. The ptf.log file contains version information for the Java Connector. The ptf.log file is located in the Connector.jar file.

See Also

Chapter 13, “Understanding jdeinterop.ini for Java Connector,” page 163

Chapter 14, “Understanding jdelog.properties File,” page 169

Running the Dynamic Java Connector

This section discusses:

- Calling a business function.
- BSFN cache.
- Transaction using the dynamic Java connector.
- OCM support for the dynamic Java connector.

Calling a Business Function

If you know the business function name and the parameters (data items) associated with the business function, you can use the dynamic Java connector to call the business function. The dynamic Java connector does not require pre-generated wrappers. This code sample shows you how to use the dynamic Java connector to call a business function:

```
import com.jdedwards.system.connector.dynamic.spec.SpecFailureException;
import com.jdedwards.system.connector.dynamic.ServerFailureException;
import com.jdedwards.system.connector.dynamic.Connector;
import com.jdedwards.system.connector.dynamic.spec.source.*;
import com.jdedwards.system.connector.dynamic.SystemException;
import com.jdedwards.system.connector.dynamic.ApplicationException;
import com.jdedwards.system.connector.dynamic.callmethod.*;

...//Declare Class

 public void execMethod() throws SpecFailureException,ServerFailureException
 {
   BSFNSpecSource specSource = null;
   // Step 1: Login
   int sessionID = Connector.getInstance().login("user", "pwd", "env","role");

   // Pre-condition: create the SpecDictionary or BSFNSpecSource
   specSource = new OneworldBSFNSpecSource(sessionID);

   // Step 2: Lookup the BSFN method from SpecDictionary or BSFNSpecSource
   BSFNMethod bsfnMethod = (BSFNMethod)specSource.getBSFNMethod
("GetEffectiveAddress");

   // Step 3: create the executable method from the BSFN metadata
   ExecutableMethod addressbook = bsfnMethod.createExecutable();
   try
```



```
    {

     // Step 4: Set parameter values
     addressbook.setValue("mnAddressNumber", "105");

     // Step 5: Execute the business function
     BSFNExecutionWarning warning = addressbook.execute(sessionID);

     // Step 6: Get return parameter values
     System.out.println("szNamealpha= " + addressbook.getValueString
("szNamealpha"));
     System.out.println("mnAddressNumber= " + addressbook.getValueString
("mnAddressNumber"));
        }
    catch (SystemException e)
    {
     //SystemException is thrown when system crash, this is a fatal
     //error and must be caught
     System.exit(1);
    }
    catch (ApplicationException e)
    {
    // ApplicationException is thrown when business function
    // execution fail, this is RuntimeException and thus can be
    // unchecked. But it is strongly recommend to catch this
    // exception
    }
    finally
    {
     //Log off and shut down connector if necessary
     Connector.getInstance().logoff(sessionID);
     Connector.getInstance().shutDown();
    }
 }
```

The dynamic Java connector permits you to use hash tables to input parameter values. This example code illustrates how to use the Hashtable class to input parameter values:

```
Map input = new Hashtable();
input.put("mnAddressNumber", String.valueOf(addressNo));
addressbook.setValues(input);
```

The dynamic Java connector permits you to use hash tables to retrieve output values. This example code illustrates how to use the Hashtable class to retrieve output values:

```
Map output = addressbook.getValues();
System.out.println("szNamealpha=" + output.getValueString("szNamealpha"));
```


BSFN Cache

The dynamic Java connector fetches a business function spec from a SpecSource (JD Edwards EnterpriseOne server or an XML repository) to create an executable method. To reduce some of the overhead for creating executable methods during run business functions, the Java connector caches the executable methods after they are created.

If OneWorldSpecSource is used as SpecSource, the dynamic Java connector gets the most current business function spec from the JD Edwards EnterpriseOne server the first time the business function is called. The cache is destructed after the connector is shutdown. This cache mechanism expedites business function execution by eliminating the overhead of retrieving the business function spec for every business function call.

The duration of the cache can be configured in the jdeinterop.ini file. You can configure the setting to balance the speed of the business function execution and the update of the business function spec.

Transaction Using the Dynamic Java Connector

You use the dynamic Java connector to do a JD Edwards EnterpriseOne transaction in either automatic or manual mode. This example code for a purchase order entry transaction shows the steps for using the dynamic Java connector in manual mode.

```
int sessionID = Connector.getInstance().login("user", "pwd", "env",
"role");
UserSession userSession = Connector.getInstance().getUserSession
(sessionID);
boolean isManulCommit;
//set isManualCommit as true or false

//Step 1: create OneWorldTransaction
OneworldTransaction transaction = userSession.createOneworldTransaction
(isManualCommit);

// Step2: create the Purchase Order Entry executable methods (such as
// poeBeginDoc, poeEditLine, poeEndDoc) from the BSFN metadata.

//Step 3: begin the transaction
transaction.begin();

//Step 4: run BSFNs in this transaction
//set poeBeginDoc input parameters (code not provided)
BSFNExecutionWarning warning = poeBeginDoc.execute(transaction);
//set poeEditLine input parameters (code not provided)
BSFNExecutionWarning warning = poeEditLine.execute(transaction);
//set poeEndDocinput parameters (code not provided)
BSFNExecutionWarning warning = poeEndDoc.execute(transaction);

//Step 5: Commit or rollback transaction
transaction.commit();
//or transaction.rollback();
```

OCM Support for the Dynamic Java Connector

You use Object Configuration Manager (OCM) to map business functions to an enterprise server so that the dynamic Java connector can access OCM to run business functions. You no longer configure the jdeinterop.ini file to define the enterprise server from which you want to execute business functions. Using OCM support should result in an increase in performance, scalability, and load balancing. The Java interoperability server distributes the processes of the Java client to various enterprise servers depending on user, environment, and role. To take advantage of dynamic Java connector OCM support:

- Configure the OCM and map the business function on different enterprise servers.
- Set OCMEnabled=true in jdeinterop.ini.
- Configure the settings in jdeinterop.ini regarding the bootstrap data source with the OCM configuration.

Ensure that OCMEnabled is set in the OCM section of the jdeinterop.ini configuration file.

See Also

Chapter 13, "Understanding jdeinterop.ini for Java Connector," page 163

Managing the User Session for the Dynamic Java Connector

This section discusses:

- User session management for the dynamic Java connector.
- Inbound XML request using the dynamic Java connector.
- Logging for the dynamic Java connector.
- Exception handling for the dynamic Java connector.

User Session Management for the Dynamic Java Connector

When the connector user successfully signs on, a valid user session is allocated to that user signon. The user session has status for two types of connector operations, one is for inbound business function calls, and the other is for outbound real-time events. The connector monitors the status of the user session and uses the time out settings in the jdeinterop.ini file to stop the user session when a time out setting has been reached. The connector looks at the these settings:

jdeinterop.ini File Section	Setting	Explanation
[CACHE]	UserSession	The maximum connector idle time for an inbound business function call.
[INTEROP]	manual_timeout	The maximum idle time for a manual transaction.
[EVENTS]	outbound_timeout	The maximum value of connector idle time for receiving outbound events.

The values for the settings are in milliseconds. A value of zero (0) indicates infinite time out. The settings are defined in the jdeinterop.ini section of this guide.

If an inbound user session times out, that user session cannot be used to execute a business function call. Likewise, if an outbound user session times out, that user session cannot be used for events. When both inbound and outbound sessions time out, the user session is removed from the connector. Since each user session has a corresponding handle in the JD Edwards EnterpriseOne server, you should explicitly call a connector API to log off the user session. The API log off releases the handle in the JD Edwards EnterpriseOne server when the user session is no longer used.

This sample code shows how to retrieve and manage a user session:

```
import com.jdedwards.system.connector.dynamic.Connector;
import com.jdedwards.system.connector.dynamic.*;
import com.jdedwards.system.connector.dynamic.ServerFailureException;

... // Declare Class
 public void execMethod() throws ServerFailureException
 {
  // Login
  int sessionID = Connector.getInstance().login("user", "pwd", "env","role");

  // Use the sessionID. If InvalidSessionException is caught, user session
is not valid any more
  //Check the status of the usersession
  UserSession session=null;
  try
  {
   session=Connector.getInstance().getUserSession(sessionID);
  }
  catch(InvalidSessionException ex)
  {
   System.out.println("Invalid user session");
  }
  if(session.isInboundTimedout())
  {
   System.out.println("User session inbound is timed out");
  }
  if(session.isOutboundTimedout())
  {
   System.out.println("User session outbound is timed out");
  }
  //Log off and shut down connector to release user session from the server
  Connector.getInstance().logoff(sessionID);
  Connector.getInstance().shutDown();
 }
```

Inbound XML Request Using the Dynamic Java Connector

You use the dynamic Java connector to send inbound synchronous XML requests (such as XML CallObject and XML List) to the JD Edwards EnterpriseOne server.

This sample code shows how to use the dynamic Java connector to execute an inbound XML request:


```
import com.jdedwards.system.xml.XMLRequest;

/... //Declare Class
        xmlInteropTest.EstablishSession(args);

    }

    public void EstablishSession(String[] args) throws Exception {
 String xmlDoc = new String();
  xmlDoc += "<?xml version='1.0' ?> <jdeRequest type='callmethod' user='user' ";
  xmlDoc += " pwd='pwd' environment='env' role='role' session='' ";
  xmlDoc += "sessionidle='1800'> </jdeRequest>";

 String requestResult;

 try {
  XMLRequest xmlRequest = new XMLRequest("E1Server", 6014, xmlDoc);
  requestResult = xmlRequest.execute();
  System.out.println("Test Successful");
 } catch (Exception e) {
  System.out.println("Error in XML request");
  System.out.println(e.getMessage());
 }
    }
```

See *JD Edwards EnterpriseOne Tools 8.96 Interoperability Guide*, "Understanding XML CallObject".

See *JD Edwards EnterpriseOne Tools 8.96 Interoperability Guide*, "Understanding XML Transaction".

See *JD Edwards EnterpriseOne Tools 8.96 Interoperability Guide*, "Understanding XML List".

Logging for the Dynamic Java Connector

Dynamic Java connector logging is built on top of Apache Open Source Project Log4j. Log4j supports five levels of logging, as listed in order of severity, from less to more:

- DEBUG
- INFO
- WARNING
- ERROR
- FATAL

The dynamic Java connector provides these APIs, located in ConnectorLog.java, to support logging information:

- public static void debug(Object source).
- public static void info(Object source).

- public static void warn(Object source).
- public static void warn(Object source, Throwable err).
- public static void error(Object source, Throwable err).
- public static void error(Object source).
- public static void fatal(Object source).
- public static void fatal(Object source, Throwable err).

Log properties (such as log file location, level of log messages to include in log file, and so on) are set in jdelog.properties. The jdelog.properties settings provide flexibility for dynamic Java connector applications to log messages. For example, you might set log level to ERROR or FATAL for a production environment or to DEBUG for a development or test environment.

See Also

Log4j Project, Apache Jakarta Project,http://logging.apache.org/log4j/docs/

Exception Handling for the Dynamic Java Connector

The dynamic Java connector error handling design provides flexibility for you to decide how to handle application-level errors. The dynamic Java connector provides these two types of exceptions to handle errors:

- ApplicationException

  This is the super class of all exceptions that result from application errors, such as InvalidConfigurationException (invalid INI settings), InvalidLoginException (invalid login), InvalidDataTypeException (invalid BSFN data type), and so on. The ApplicationException is a runtime exception. It is up to the client program to catch this type of exception.

- SystemException

  This is the super class of all exceptions that result from system errors, such as ServerFailureException (server down or connection failure), BSFNLookupFailureException (unable to find BSFN information in JD Edwards EnterpriseOne tables), and SpecFailureException (unable to connect to Spec Source). It is up to the client program to catch this type of exception.

Using Sample Applications

This section discusses:

- Sample applications
- Setting up sample applications
- Running the sample applications

Sample Applications

These applications are shipped with the dynamic Java connector in their Java source form:

Application	Description
Address Book	Queries an AddressBook entry.
Events	Subscribes to events.
Manual Commit	Performs a local transaction using a Purchase Order Entry application.
Purchase Order	Enters a purchase order.
Sales Order	Enters a sales order.

Before you use the sample applications:

- Create a directory for the sample applications (for example, C:\connectorsamples).
- Install a Java Development Kit (JDK) version 1.4 or higher. Be sure to install a full JDK and not the Java Runtime Environment (JRE).

  See Chapter 9, "Working with the Dynamic Java Connector," Installing the Dynamic Java Connector, page 97.
- Set the JAVA_HOME environment variable to the JDK parent directory.
- Configure the jdeinterop.ini, jdelog.properties, and jdbj.ini files and place the files in the directory you created for the sample applications (for example, C:\connectorsamples).

Note. You can download the JDK from the Sun Microsystems website (java.sun.com/j2se).

Setting Up Sample Applications

The sample applications are shipped in their Java source form, which provides the usage of the dynamic Java connector API. You must set up these sample applications in the environment before you can run them. Use these steps to set up the sample applications:

1. Locate the connector_samples_src.jar and connectorsamples.zip files.

   These files are on the JD Edwards EnterpriseOne Java Server CD, under the system/classes/samples directory.
2. Unzip the entire contents of the connector_samples_src.jar file and connectorsamples.zip into the directory you created (for example, C:\connectorsamples).

   The .jar file is a traditional .zip file with the Java .jar extension. The .jar file contains all of the sample application source files (.java files). All of the .jar files that you need for both setting up and running the sample applications are in the system/classes directory on the JD Edwards EnterpriseOne Java Server CD.
3. Open each bat file in the samples directory and change the value of JAVA_HOME to the path where JDK is installed on your system.
4. Configure the jdeinterop.ini, jdelog.ini, and jdbj.ini files and place them in the samples directory.

   You can use .tmpl files as a guide for doing this.

Running the Sample Applications

To run each application, run the .bat file for that application.

Sample Application	Bat File name
Address Book	runDynConAddressBook.bat
Events	runDynConNewEventDriver.bat
Manual Commit	runDynConPOEManualCommit.bat
Purchase Order	runDynConPOE.bat
Sales Order	runDynConSOE.bat

Note. If you are running on a non-windows platform, you can open the bat file that corresponds to the sample application that you want to use in a text editor and copy the JAVA command in the bat file. This command can then be run from the console of your platform. The correct version of JAVA must be in the system path for you to run the application.

CHAPTER 10

Understanding the Java Connector

This chapter provides an overview of the Java connector and discusses how to:

- Design the Java connector.
- Install the Java connector.
- Run the Java connector.
- Manage the user session for the Java connector.
- Use exception handling for the Java connector.

Note. If this is the first implementation of a Java connector, it is suggested that you consider the dynamic Java connector instead of the Java connector. The functionality is the same. The advantage of implementing the dynamic Java connector is that you are not required to generate wrappers

---

Java Connector and JD Edwards EnterpriseOne

A business function is a logical collection of C functions and their associated data structures grouped together to produce a unit of work. JD Edwards EnterpriseOne Java objects are wrappers, implemented in Java, around these business functions and data structures.

The method that a Java wrapper provides has a one-to-one correspondence with business functions. Because all methods must be defined in a Java class, a library must be defined in the corresponding iJDEScript file.

For example, if library A contains business function B550001, and within this business function two C functions exist, named foo1 and foo2, with data structures for each function named DS1 and DS2, then the corresponding Java class would be as follows:

```
Public class A
{
   public int foo1(DS 1 param, OneWorldInterface ow,
         Connector c, int handle)
   {
   0
   }
   public int foo2(DS2 param, OneWorldInterface ow,
         Connector c, int handle)
   {
   0
   }
   public DS1 Createfoo1ParameterSet()
   {
   0
```

```
    }
    public DS2 Createfoo2ParameterSet()
    {
    0
    }
}
```

For each business function X, a method CreateXParameterSet exists in the class that returns a class for the data structure used by the business function.

Each data structure has a corresponding Java class, and each element in the data structure has a *get* and a *set* method. For example, if DS1 has element A as a char, the DS1 Java class is as follows:

```
Public class DS1
{
    public void setA()
    {
        ...
    }
    public char getA()
    {
        ...
    }
}
```

The data structure can contain two kinds of compound objects, JDEDate and JDEMathnumeric, in addition to the primitive data types. The two Java classes JDEDate and JDEMathnumeric are defined respectively.

JDEDate

This table provides JDEDate methods and a description of the method:

Method	Description
JDEDate()	Construct a JDEDate.
getDay()	Get the day of the date.
getMonth()	Get the month of the date.
getYear()	Get the year of the date.
setDay(short)	Set the day of the date.
setMonth(short)	Set the month of the date.
setYear (short)	Set the year of the date.

JDEMathNumeric

This table shows the JDEMathNumeric methods and provides a description of each method:

Method	Description
getValue()	Return the value as a string (for example, -12345.6789).
setValue(String strValue)	Set the value from a string (for example, -12345.6789).
getCurrencyDecimals()	Get the currency decimal positions.
setCurrencyDecimals(int aValue)	Set the currency decimal positions.
getCurrencyCode()	Get the currency code.
setCurrencyCode(String aValue)	Set the currency code.
getDecimalPosition()	Get the decimal position.
isNegative()	Test if the value is negative.
reset()	Reset all the internal values.

To set the value of a member in a MathNumeric type in a data structure, use the method setValue(String) in JDEMathNumeric class. For example, if mnAddressBook is a member in the data structure, then a class should exist for the data structure with the public method getmnAddressBook, which returns a JDEMathNumeric object. Then you use `DS.getmnAddressBook().setValue(1)` to set the mnAddressBook value to 1 in the data structure.

Designing the Java Connector

This section covers considerations for designing the Java connector solution and discusses:

- GenJava
- Java versioning
- GenJava client environment

GenJava

The JD Edwards EnterpriseOne system provides a Java generation tool, GenJava, that you run to expose business functions through Java. A system administrator usually runs GenJava.

When you run GenJava, you specify a library of business functions to wrap, for example CAEC. GenJava creates Java class files for all the business functions and associated data structures. GenJava also compiles the business functions, generates Java docs, and packages them to two JAR files, one for Java classes and one for Java documents. For example, if the library is JDEAddressBook, you see JDEAddressBookInterop.jar and JDEAddressBookInteropDoc.jar in either the B9\system\classes directory or any directory redirected by GenJava.

GenJava Client Environment

When you set up a client environment for GenJava, ensure the PATH environment variable and the CLASSPATH environment variable are set up correctly.

PATH

<bin directory for JDK>

Example: c:\jdk1.2.2\bin

CLASSPATH

<Directory where JD Edwards EnterpriseOne is located>\System\classes\base_JAR.jar

<Directory where JD Edwards EnterpriseOne is located>\System\classes\jdeNet_JAR.jar

<Directory where JD Edwards EnterpriseOne is located>\System\classes\system_JAR.jar

<Directory where JD Edwards EnterpriseOne is located>\System\classes\connector.jar

<Directory where JD Edwards EnterpriseOne is located>\System\classes\xalan.jar

<Directory where JD Edwards EnterpriseOne is located>\System\classes\xerces.jar

Java Versioning

Business object wrappers that are generated for one environment might not be compatible with another environment. Versioning prevents you from creating Java business objects unless the environment used at logon is the same as the environment used to generate the wrappers or the environment is compatible with the business objects. You can use the Java Wrapper Version Checker (CheckVer) to verify that business object wrappers are compatible with new environments.

Migrating from Previous Releases

Previously generated business object wrappers are compatible with the new versioning code; you do not need to regenerate them. However, in order to use them, CheckVer must be run, even for the environment used to create the wrappers. The repository setting in the [INTEROP] section of the ini file must point to the directory containing the jar files of generated business object wrappers. For example:

```
[INTEROP]
repository=c:\foo\bar\repository
```

The repository directory should contain only jar files for generated business object libraries.

Java Connector Static and Dynamic Modes

A Java interoperability client can be configured statically or dynamically. Static mode is the normal mode of operation and should be used by most client code. Dynamic mode is better suited for developing tools based on Java interoperability. The two modes can be used simultaneously in the same process. The granularity is at the business object library (jar file) level. No matter which mode is used, it is necessary for the jar files to be placed in the repository directory.

To use static mode for a given business object library, ensure that the jar file is in both the classpath and repository directory for the client process.

To use dynamic mode for a given business object library, ensure that the jar file is in the repository directory but not in the classpath. Dynamic mode is for Java interoperability clients with client code that has no direct use of the business objects. In dynamic mode, business objects may only be used by the classes in the java.lang.reflect package. Dynamic mode enables client code to refresh, add, or remove business object libraries while in operation. These operations are accomplished using the methods in the OneWorldVersion class (for example, generate a new business object library (or regenerate an existing library) using GenJava). Use the CheckVer tool to establish the compatible environments for the business objects in the library. Add the jar file to the repository directory. Finally, the client code must instantiate a OneWorldVersion object, and call the refreshLibrary method. To remove a business object library, remove it from the repository and call the refreshLibrary method.

After a library is refreshed, all newly created business objects use the new definition. Business objects created before the refresh use the old definition. No limit exists for the number of simultaneous business object library versions. The old library definitions remain in the virtual machine until no more references to the old business objects exist, which can significantly affect memory use in the virtual machine.

Using the Java Wrapper Version Checker (CheckVer)

CheckVer is a Java class and should not be confused with the CheckVer.exe that is a part of the COM interoperability solution. You run CheckVer to verify whether a previously generated Java business object library is compatible with another environment. Typically, the system administrator performs this task. The XML files generated by GenJava are the signatures of the objects generated against specific JD Edwards EnterpriseOne environments. These XML files can be used with CheckVer to verify that the wrappers in a previously generated jar file are compatible with the environment.

When you introduce a new JD Edwards EnterpriseOne environment, you run GenJava against the new environment by using the /XMLOnly option. You also use the iJDEScript that you used to generate the wrappers to generate XML signature files for the objects in the new environment. Run CheckVer with the new XML files and previously generated jar files to verify that the new environment is compatible with the wrappers. CheckVer updates the jar file according to the result of the compatibility test. A Java client using the jar file can be dynamically updated to the new compatibility information, using the OneWorldVersion interface. If the new environment is incompatible, the client is not allowed to create business objects with the new environment.

Running CheckVer (GenJava)

CheckVer takes two arguments, the jar file name and the XML file name. CheckVer requires that the connector.jar, base_JAR.jar, jdeNet_JAR.jar, system_JAR.jar, xalan.jar, and xerces.jar files be in the CLASSPATH. This can be done either with the CLASSPATH environment variable or from the command line.

Syntax

Java com.jdedwards.system.connector.CheckVer [*jarfile*] [*xmlfile*]

Example

Java com.jdedwards.system.connector.CheckVer *JDEAddressBookInterop.jar JDEAddressBook.xml*

Installing a Java Connector

These steps illustrate how to install Java connector components so that you can run a Java connector application.

1. Copy these files from the enterprise server to a directory on the desired machine. For example, copy these files to C:\JDEdwards\Interop on the machine:
   - connector.jar
   - log4j.jar
   - base_JAR.jar
   - jdeNet_JAR.jar
   - system_JAR.jar
   - jdbjBase_JAR.jar
   - jdbjInterfaces_JAR.jar
   - xalan.jar
   - xerces.jar
   - jdeinterop.ini
   - jdelog.properties
   - JDBC driver (you need to get JDBC driver from the vendor)
2. Add these files to the CLASSPATH:
   - connector.jar
   - log4j.jar
   - base_JAR.jar
   - jdeNet_JAR.jar
   - system_JARj.jar
   - jdbjBase_JAR.jar
   - jdbjInterfaces_JAR.jar
   - xerces.jar
   - JDBC driver
3. Add the path where the jdelog.properties and jdeinterop.ini files are located into CLASSPATH.
4. Create a separate repository directory for business object.jar files.
5. Run GenJava on the client machine and copy the output jar file (for example, JDEAddressBook.jar) to this directory.
6. Depending on whether you want the library in static mode or dynamic mode, put the business object.jar file in the CLASSPATH.

Note. The ptf.log file contains version information for the Java Connector. The ptf.log file is located in the connector.jar file.

Running the Java Connector

This section covers runtime considerations for the Java connector and discusses:

- Using GenJava

- Using GenJava output
- Transactions Using the Java connector

Using GenJava

The Java generator tool, GenJava, provides access to business functions by generating Java interfaces for business functions. GenJava includes these components:

- GenJava.exe
- Emitter framework
- JDEIDAJavaEmitter.dll

You use iJDEScript scripting language to script code generation activities when you use GenJava.

Running GenJava

You run GenJava from the command line. There are several options available for generation. GenJava is located in <install>\system\bin32.

Syntax

GenJava [options] [libraries]

Options

You can use these options when running GenJava:

Option	Description
/?	Lists the options available for generation.
/Cat <category>	Generates only <category> function wrappers. Supports these categories: /'1/' - Master Business Functions /'2/' - Major Business Functions /'3/' - Minor Business Functions /'-/' - Uncategorized Business Functions
/Cmd *	Processes code generation commands from the console.
/Cmd <filename>	Processes code generation commands from <filename>.
/Compiler <file>	Uses <file> to compile Java files.
/D name value	Defines a macro value.
/EnvironmentID <env>	Uses <env> to sign on to JD Edwards EnterpriseOne.
/ListLibraries	Lists the available libraries that you can use for GenJava.
/MsgFile <file>	Provides GenJava with the file name to log messages produced by GenJava during the generation process; for example, messages.log.

Option	Description
/NoBSFN	Tells GenJava not to create wrappers for business functions. This option is for generating parameter sets only.
/Out <path>	Provides GenJava with the directory (path) in which to place the output files; for example, C:\winnt\system32.
/Password <password>	Provides GenJava with the password with which you want to sign on to JD Edwards EnterpriseOne.
/Role	Provides GenJava with the role with which you want to sign on to JD Edwards EnterpriseOne.
/TempOut <path>	Provides GenJava with the directory (path) in which to place temporary files needed for the build process; for example, C:\temp.
/UserID <userid>	Provides GenJava with the user name that you use to sign on to JD Edwards EnterpriseOne.
/XMLOnly	Generates only the XML file.

You can also use GenJava by running it with a JDEScript file, such as:

```
GenJava /cmd AddressBook.cmd
```

This command prompts a sign-in window for you to enter the user ID, password, role, and environment. The AddressBook.cmd is:

```
define library JDEAddressBook
login
library JDEAddressBook
library JDEAddressBook
interface AddressBook
interface AddressBook
import B0100031
import B0100019
import B0100032
import B0100002
import B0100033
build
logout
```

GenJava generates the wrappers in Java for all business functions imported in the script file.

Generate Java Wrappers

This command generates Java wrappers for Category 1 business functions in the CAEC library:

GenJava /Cat 1 /UserID Devuser1 /Password Devuser1 /EnvironmentID ADEVHP02 CAEC

You must use the correct information (including user ID, password, role, and environment) to log on to JD Edwards EnterpriseOne.

Using GenJava Output

The output for GenJava produces fully functional Java objects based on the library you use to generate wrappers. GenJava packages these objects in a single jar file such as XXXXInterop.jar or XXXXInteropDoc.jar, where XXXX is the library name defined in the script file or from the command line. For example, JDEAddressBookInterop.jar is created for the AddressBook.cmd. The default location for the jar file is under B9/System/classes, but it can be somewhere else if you run GenJava using /Out value. This jar file must be deployed to the machine that uses those wrappers. To import any wrapper object and class, the jar file must be added to the CLASSPATH. Because you are interacting with JD Edwards EnterpriseOne, these components, connector.jar, base.JAR.jar, jdeNet_JAR.jar, system_JAR.jar, and jdeinterop.ini file, must be deployed to the machine.

XXXXInteropDoc.jar is the compressed format of all the Java documents (html files) for all the classes generated by GenJava.unjar. You can also unzip the jar file to see the APIs that can be called in these classes.

All Java client applications must:

1. Initialize a com.jdedwards.system.connector.Connector.
2. Sign in to JD Edwards EnterpriseOne using a valid user ID, password, role, and environment name. The environment must be valid on the JD Edwards EnterpriseOne server.
3. Get the OneWorldInterface object reference by calling Connector.CreateBusinessObject with an object name, such as Connector::OneWorldInterface.
4. Get the object reference for the wrapper for the business function generated by GenJava, for example AddressBook. The object name passed into Connector.CreateBusinessObject should be Library (Java package) Name:Object Name, such as JDEAddressBook:AddressBook.
5. Call CreateXXXParameterSet on the wrapper object for any data structure XXX.
6. Set the needed value in the data structure.
7. Call the business function with the data structure variable as a parameter. Check the return value. The return value can be one of these:

   Successful = 0

   Warning = 1

   Error = 2
8. Process the data returned by the business function.
9. Disconnect from JD Edwards EnterpriseOne.

These examples illustrate how to use a generated Java business function wrapper in a Java application.

```
import com.jdedwards.system.connector.*;
import com.jdedwards.application.interop.jdeaddressbook.*;

... Declare Class
 {
 Connector connectorProxy = null;
 OneWorldInterface ow;
 AddressBook ab;
 D0100033 ds;
 int sessionID=0;
 connectorProxy = new Connector();
 try
```



```
 {
 // sessionID = connectorProxy.login("user", "pwd", "role");
  sessionID = connectorProxy.Login("user", "pwd", "role");
  System.out.println("Log in successfully");
 }
 catch (reject r)
 {
  System.out.println("got reject exception");
  String s = r.reason;
  System.out.println(s);
  System.exit(1);
 }
 catch (Exception e)
 {
  System.out.println("got other exception");
  e.printStackTrace();
  System.exit(1);
 }
 try
 {
  ow = (OneWorldInterface)connectorProxy.CreateBusinessObject
("Connector::OneWorldInterface", sessionID);
  System.out.println("got OneWorldInterface");
 }
 catch (reject r)
 {
  String s = r.reason;
  System.out.println(s);
  return;
 }
//create AddressBook object
 try
 {
  ab = (AddressBook)connectorProxy.CreateBusinessObject
("JDEAddressBook::AddressBook", sessionID);
  System.out.println("got AddressBook");
 }
 catch (reject r)
 {
  String s = r.reason;
  System.out.println(s);
  return;
 }
// get data structure D0100033
 ds = ab.CreateGetEffectiveAddressParameterSet();
// set addressbook number value in D0100033
 ds.getmnAddressNumber().setValue("1");
// get address information
 int i = 0;
 try
```

```
 {
  i = ab.GetEffectiveAddress(ds, ow, connectorProxy, sessionID);
 }
 catch (reject e)
 {
  System.out.println(e.reason);
 }
 if (i!=2)
 {
  String alphaname = ds.getszNamealpha();
  String address = ds.getszAddressLine1();
  String zipcode = ds.getszZipCodePostal();
  String city = ds.getszCity();
  String county = ds.getszCountyAddress();
  String state = ds.getszState();
  String country = ds.getszCountry();
  System.out.println("ALpha Name "+alphaname);
  if (i==1)
  {
   System.out.println("warning count is"+ow.GetWarningCount());
   for ( int j = 0; j<ow.GetWarningCount(); j++)
   {
    String s = ow.GetWarningAt(j);
    System.out.println("warning" + j +";"+ s);
   }
  }
 }
 else
 {
  for (int j = 0; j<ow.GetErrorCount(); j++)
  {
   String s = ow.GetErrorAt(j);
   System.out.println("error" + j + ";" + s);
  }
  System.out.println("BSFN error");
//log off
  connectorProxy.Logoff(1);
 } // end main

}
```

Transactions Using the Java Connector

Transactions are a way to update the JD Edwards EnterpriseOne database. You can use the Java connector to do a transaction in either auto mode or manual mode. When you use auto transaction mode, the transaction is immediately committed after the business function call is completed. The transaction is set to the auto commit mode by the system. When you use manual transaction mode, the transaction is started by explicitly calling BeginTransaction in OWInterface, and the transaction is committed (or rolled back) by calling Commit (or Rollback) in OWInterface.

Note. The JD Edwards EnterpriseOne transaction is not really a two-phase commit. You need to manually roll back the transaction when the commit statement is reached.

This example shows a basic manual commit transaction:

```
import com.jdedwards.system.connector.dynamic.ApplicationException;
import com.jdedwards.system.connector.dynamic.SystemException;
import com.jdedwards.system.connector.dynamic.sample.DynConApplication;
import com.jdedwards.system.connector.dynamic.callmethod.ExecutableMethod;
import com.jdedwards.system.connector.dynamic.callmethod.BSFNExecutionWarning;
import com.jdedwards.system.connector.dynamic.spec.source.BSFNSpecSource;

import java.util.*;

/
... Declare Class
{
    private boolean isBegindocCalled = false;
    private boolean isEditlineCalled = false;
    private ExecutableMethod soeBeginDoc = null;
    private ExecutableMethod soeEditLine = null;
    private ExecutableMethod soeEndDoc = null;
    private ExecutableMethod soeClearWF = null;


    public SalesOrderEntryApplication(int sessionID, BSFNSpecSource specSource) {
        super(sessionID, specSource);
    }


    public BSFNExecutionWarning executeBeginDoc(Map inputParams,
Map outputParams) throws SystemException {
        soeBeginDoc = getBSFNMethod("F4211FSBeginDoc");
        // check the necessary settings

        // set the user define values
        soeBeginDoc.setValues(inputParams);

        // set default value
        soeBeginDoc.setValue("cCMDocAction", "A");
        soeBeginDoc.setValue("cCMProcessEdits", "1");
        soeBeginDoc.setValue("cCMUpdateWriteToWF", "2");
        soeBeginDoc.setValue("szCMProgramID", "CORBA");
        soeBeginDoc.setValue("szCMVersion", "ZJDE0001");
        soeBeginDoc.setValue("cMode", "F");
        soeBeginDoc.setValue("cRetrieveOrderNo", "1");
        soeBeginDoc.setValue("szCMComputerID", getComputerName());
        if (isEmpty(inputParams.get("szOrderType"))) {
            soeBeginDoc.setValue("szOrderType","SO");
```



```
        }

        if (isEmpty(inputParams.get("jdOrderDate"))) {
            soeBeginDoc.setValue("jdOrderDate", getCurrentDate());
        }

        BSFNExecutionWarning warning = soeBeginDoc.execute(sessionID);
        setOutput(outputParams,soeBeginDoc.getValueStrings());
        isBegindocCalled= true;
        return warning;
    }


    public BSFNExecutionWarning executeEditLine(Map inputParams,
Map outputParams) throws SystemException {
        // Edit Line
        if (!isBegindocCalled) {
            throw new ApplicationException("BeginDoc must be called
before editline");
        }
        soeEditLine = getBSFNMethod("F4211FSEditLine");

        // set user input values
        soeEditLine.setValues(inputParams);

        // set default values
        soeEditLine.setValue("mnCMJobNo", soeBeginDoc.getValue("mnCMJobNumber"));
        soeEditLine.setValue("mnOrderNo", soeBeginDoc.getValue("mnOrderNo"));
        soeEditLine.setValue("szBusinessUnit", soeBeginDoc.getValue
("szBusinessUnit"));
        soeEditLine.setValue("szCMComputerID", soeBeginDoc.getValue
("szCMComputerID"));
        soeEditLine.setValue("cCMWriteToWFFlag", "2");
        soeEditLine.setValue("szOrderType", soeBeginDoc.getValue("szOrderType"));

        BSFNExecutionWarning warning = soeEditLine.execute(sessionID);
        setOutput(outputParams,soeEditLine.getValueStrings());
        isEditlineCalled = true;
        return warning;
    }


    public BSFNExecutionWarning executeEndDoc(Map inputParams,
Map outputParams) throws SystemException {
        if (!isBegindocCalled) {
            throw new ApplicationException("BeginDoc must be called before
EndDoc");
        }
        soeEndDoc = getBSFNMethod("F4211FSEndDoc");
        soeEndDoc.setValues(inputParams);
```



```
        soeEndDoc.setValue("mnCMJobNo", soeBeginDoc.getValue("mnCMJobNumber").
toString());
        soeEndDoc.setValue("mnSalesOrderNo", soeBeginDoc.getValue("mnOrderNo").
toString());
        soeEndDoc.setValue("szOrderType", soeBeginDoc.getValue("szOrderType"));
        soeEndDoc.setValue("szCMComputerID", getComputerName());
        soeEndDoc.setValue("cCMUseWorkFiles", "2");

        BSFNExecutionWarning warning = soeEndDoc.execute(sessionID);
        isBegindocCalled = false;
        isEditlineCalled = false;
        setOutput(outputParams,soeEndDoc.getValueStrings());
        return warning;
    }

    public BSFNExecutionWarning executeClearWF(Map inputParams,
Map outputParams) throws SystemException {
        if (isBegindocCalled) {
            soeClearWF = getBSFNMethod("F4211ClearWorkFile");
            soeClearWF.setValues(inputParams);
            soeClearWF.setValue("cClearDetailWF", "2");
            if (isEditlineCalled) soeClearWF.setValue("cClearHeaderWF", "2");
            soeClearWF.setValue("mnJobNo", soeBeginDoc.getValue("mnCMJobNumber"));
            soeClearWF.setValue("szComputerID", getComputerName());
            soeClearWF.setValues(inputParams);
            BSFNExecutionWarning warning = soeClearWF.execute(sessionID);
            setOutput(outputParams,soeClearWF.getValueStrings());
            return warning;
        }
        return null;
    }
```

Using BHVRCOM through the Java Connector

You use the BHVRCOM structure to control the execution of business functions. You use the Java connector to call methods in the OWInterface class to set and pass the BHVRCOM fields to business functions on the server. This table shows the business function methods and the BHVRCOM fields:

Business Function Method	BHVRCOM Field
setBOBMode(int bobMode)	IBobMode
setAPPName(StringaName)	szApplication
setUserName(String aName)	szUser
setDatabaseChanged(Boolean value)	bDataBaseChange

This Java code demonstrates how to query the IBHVRCOM interface and pass values to business functions:

```
...
   ow = (OneWorldInterface)
connectorProxy.CreateBusinessObject("Connector::OneWorld Interface", l);
ab=(AddressBook)connectorProxy.CreateBusinessObject
("JDEAddress Book::Address Book", l);
ds.getmnAddressNumber().setValue("1");
ow.setAppName("AddressbookApp");
ow.setBOBMode(8);
ow.setUserName("Java Connector");
ow. SetDatabaseChanged(false);
   i = ab.GetEffectiveAddress(ds, ow, connectorProxy, l);
...
```

OCM Support for the Java Connector

You use Object Configuration Manager (OCM) to map business functions to an enterprise server so that the Java connector can access OCM to run business functions. You no longer configure the jdeinterop.ini file to define the enterprise server from which you want to execute business functions. Using OCM support should result in an increase in performance, scalability, and load balancing. The Java interoperability server distributes the processes of the Java client to various enterprise servers depending on user, environment, and role. To take advantage of Java connector OCM support:

- Use a B9 or later version of GenJava to regenerate the business wrapper function.
- Configure the OCM and map the business function on different enterprise servers.
- Set OCMEnabled=true in jdeinterop.ini.
- Configure the settings in jdeinterop.ini regarding the bootstrap data source with the OCM configuration.

Ensure that these settings in the jdeinterop.ini configuration file are set:

jdeinterop.ini File Section	Required Settings
OCM	OCMEnabled
JDBj-BOOTSTRAP SESSION	user, password, environment, and role
JDBj-BOOTSTRAP DATA SOURCE	name, databaseType, server, database, serverPort, physicalDatabase, library, owner
[JDBj-JDBC DRIVERS]	ORACLE, iSeries, SQLSERVER, UDB
[JDBj-ORACLE]	tns

Managing the User Session for the Java Connector

This section provides an overview of managing the user session for the Java connector and discusses inbound XML requests using the Java connector.

Understanding User Session Management for the Java Connector

When the connector user successfully signs on, a valid user session is allocated to that user signon. The user session has status for two types of connector operations: one for inbound business function calls and the other for outbound real-time events. The connector monitors the status of the user session, and uses the timeout settings in the jdeinterop.ini file to stop the user session when a timeout setting has been reached. The connector looks at these settings:

jdeinterop.ini File Section	Setting	Explanation
[CACHE]	UserSession	The maximum connector idle time for an inbound business function call.
[INTEROP]	manual_timeout	The maximum idle time for a manual transaction.
[EVENTS]	outbound_timeout	The maximum value of connector idle time for receiving outbound events.

The value for the settings is in milliseconds. A value of zero (0) indicates infinite timeout. The settings are defined in the jdeinterop.ini section of this guide.

If an inbound user session times out, that user session cannot be used to execute a business function call. Likewise, if an outbound user session times out, that user session cannot be used for events. When both inbound and outbound sessions time out, the user session is removed from the connector. Since each user session has a corresponding handle in the JD Edwards EnterpriseOne server, it is *highly* recommended that you explicitly call a connector API to log off the user session to release the handle in the JD Edwards EnterpriseOne server when the user session is no longer used.

This sample codes shows how to retrieve and manage a user session:

```
import com.jdedwards.system.connector.dynamic.Connector;
import com.jdedwards.system.connector.dynamic.*;
... //Declare Class
 {
  // Login
  int sessionID = Connector.getInstance().login("user", "pwd", "env","role");
  // Use the sessionID. If InvalidSessionException is caught, user session is
not valid any more
  //Check the status of the usersession
  UserSession session=null;
  try
  {
   session=Connector.getInstance().getUserSession(sessionID);
  }
  catch(InvalidSessionException ex)
  {
   System.out.println("Invalid user session");
   if(session.isInboundTimedout())
   {
    System.out.println("User session inbound is timed out");
   }
```



```
  if(session.isOutboundTimedout())
  {
   System.out.println("User session outbound is timed out");
  }
  Connector.getInstance().logoff(sessionID);
  Connector.getInstance().shutDown();
 }
}
```

Inbound XML Request Using the Java Connector

You use the Java connector to send inbound synchronous XML requests (such as XML CallObject and XML List) to the JD Edwards EnterpriseOne server. The Java connector has an API that it calls to send XML documents to JDENET.

This example code shows how to use the Java connector to execute an inbound XML request:

```
Connector conn = new Connector();
//login into OW
String xmlDoc;
//or byte[] xmlDoc
//Load a String or byte[] into xmlDoc;

String requestResult = conn.executeXMLRequest(xmlDoc);
//handle requestResult.
```

See *JD Edwards EnterpriseOne Tools 8.96 Interoperability Guide*, “Understanding XML CallObject”.

See *JD Edwards EnterpriseOne Tools 8.96 Interoperability Guide*, “Understanding XML Transaction”.

See *JD Edwards EnterpriseOne Tools 8.96 Interoperability Guide*, “Understanding XML List”.

Using Exception Handling for the Java Connector

This section provides an overview for exception handling for the Java connector and discusses:

- Fatal exception
- Recoverable exception
- Reject
- Exception details

This section also provides sample code for Java connector exception handling.

Understanding Exception Handling for the Java Connector

When you run the Java connector or the GenJava tool, the program might encounter a condition that causes unexpected results or system failure. When the program does not perform as expected, an error occurs; or, using Java terminology, an exception is thrown. In Java, the system, classes, and programs can throw exceptions. You can write code to catch exceptions. Catching an exception involves dealing with the exception conditions so that the program will not crash.

All exceptions in the connector and GenJava code inherit from the reject class. The program needs to catch only the reject exception conditions for the methods that throw exceptions. To help minimize manual intervention, the FatalException and the RecoverableException classes were created so that you can provide a recovery action in the program for some exceptions.

Fatal Exception

FatalException class conditions are unlikely or impossible to resolve without manual intervention. If you catch fatal exception conditions in the program, you can include a string message that indicates the condition that occurred. You use the getMessage method from the java.lang.Throwable class to retrieve fatal exception messages from the program. The system uses the INTEROP category to log fatal exception conditions to the jas.log file.

Recoverable Exception

You can provide the capability for the system to possibly resolve an exception condition by catching RecoverableException (and children) class conditions in the program. The children of recoverable exception conditions indicate through their class names the category of the exception and include a sting message in the constructor to provide more exception details. You use the getMessage method from the java.lang.Throwable class to retrieve recoverable exception messages from the program. The system uses the INTEROP category to log recoverable exception conditions to the jasdebug.log file. You can clear recoverable exception messages through the DEBUG flag in the jdeinterop.ini file. The flag is either true or false.

Reject

The method signature for each of the methods listed in this table indicates that the method only throws reject, even though the exceptions thrown in each method's code are children of the reject class. Even if you decide to catch all of the exceptions listed in Exception Details table (which follows), you also need to catch *reject* as the last in the series of connector-related catch statements because of the *throws* clause in the method signature.

Exception Details

The methods that throw exceptions in each of the main public classes of the connector (Connector, OneWorldInterface, EventSource, and GenJava-created business object code) are detailed in this table. The information in this table is also available in the Javadoc for the connector, which is in the ConnectorDoc.jar file.

Class	Method	Exception	Condition	Possible Action
Connector	Login	CallObjectRetryException	The error code returned by CallObject is TIMEOUT or RETRY_NEEDED	Retry Login method

Class	Method	Exception	Condition	Possible Action
N/A	N/A	CallObjectIgnoreException	The error code returned by CallOjbect is NOERROR, ALREADY_ EXECUTED, or BAD_ERRORPACKETS	Ignore this exception
N/A	N/A	FatalException	The error code returned by CallObject is any other error code	*
N/A	CreateBusiness Object	NotLoggedInException	The user is not currently logged in to JD Edwards EnterpriseOne	Log in through Connector class
N/A	N/A	FatalException	A Java reflection exception is thrown or the JD Edwards EnterpriseOne environment is not in sync with the business function wrapper	*
OneWorld Interface	GetNextError	NoMoreDataException	Error index reaches the end of the array	End the loop searching for the next error
N/A	GetNextWarning	NoMoreDataException	Warning index reaches the end of the array	End the loop searching for the next warning
N/A	Commit	InvalidMethodCall Exception	This method is called before PrepareToCommit() is called	Call the PrepareToCommit() method
N/A	N/A	CallObjectRetryException	The error code returned by CallObject is TIMEOUT or RETRY_NEEDED	Retry Commit method
N/A	N/A	CallObjectIgnoreException	The error code returned by CallObject is NOERROR, ALREADY_ EXECUTED, or BAD_ERRORPACKETS	Ignore this exception
N/A	N/A	FatalException	The error code returned by CallObject is any other error code	*
N/A	Rollback	CallObjectRetryException	The error code returned by CallObject is TIMEOUT or RETRY_NEEDED	Retry Rollback method
N/A	N/A	CallObjectIgnoreException	The error code returned by CallObject is NOERROR, ALREADY_ EXECUTED, or BAD_ERRORPACKETS	Ignore this exception

Class	Method	Exception	Condition	Possible Action
N/A	N/A	FatalException	The error code returned by CallObject is any other error code	*
N/A	PrepareToCommit	CallObjectRetryException	The error code returned by CallObject is TIMEOUT or RETRY_NEEDED	Retry PrepareToCommit method
N/A	N/A	CallObjectIgnoreException	The error code returned by CallObject is NOERROR, ALREADY_ EXECUTED, or BAD_ERRORPACKETS	Ignore this exception
N/A	N/A	FatalException	The error code returned by CallObject is any other error code	*
N/A	ExecuteBSFN	NotLoggedInException	The user is not currently logged in to JD Edwards EnterpriseOne	Log in through Connector class
N/A	N/A	CallObjectRetryException	The error code returned by CallObject is TIMEOUT or RETRY_NEEDED	Retry ExecuteBSFN method
N/A	N/A	CallObjectIgnoreException	The error code returned by CallObject is NOERROR, ALREADY_ EXECUTED, or BAD_ERRORPACKETS	Ignore this exception
N/A	N/A	FatalException	The error code returned by CallObject is any other error code	*
Event Source	EventSource (Constructor)	FatalException	The connector cannot listen on the given port	*
N/A	addListener	NotLoggedInException	The user is not currently logged in to JD Edwards EnterpriseOne	Log in through Connector class
N/A		FatalException	The subscription fails	*
N/A	removeListener	NotLoggedInException	The user is not currently logged in to JD Edwards EnterpriseOne	Log in through Connector class
N/A		FatalException	The unsubscription fails	*
N/A	updateSession	NotLoggedInException	The user is not currently logged in to JD Edwards EnterpriseOne	Log in through Connector class

Class	Method	Exception	Condition	Possible Action
N/A	getEventTemplate	NotLoggedInException	The user is not currently logged in to JD Edwards EnterpriseOne	Log in through Connector class
N/A	N/A	FatalException	A JdeNetException is thrown	*
N/A	getEventTypes	NotLoggedInException	The user is not currently logged on to JD Edwards EnterpriseOne	Log in through Connector class
N/A	N/A	FatalException	A JdeNetException is thrown	*
GenJava-created Data Structures	setString <parameter> methods	StringTooLongException	The value set for the parameter is too long	Reset the parameter using a shorter length

For FatalException conditions, you can send the exception message, which can be retried by using the getMessage method, to the system administrator. Alternatively, you can prompt the system administrator to look in the jas.log file for more details about the exception. It is unlikely that the program can recover associated system or connector errors during runtime.

Example: Java Connector Exception Handling Sample Code

This code illustrates some of the features of the enhanced connector exception handling. The bold-faced items indicate specific exception-handling code.

```
import com.jdedwards.system.connector.*;
import com.jdedwards.application.interop.jdeaddressbook.*;

... //Declare Class
{
  if (args.length != 1)
  {
   System.out.println("Must supply a city to query for AddressBook");
   System.exit(-1);
   Connector connectorProxy = null;
   OneWorldInterface ow = null;
   AddressBook ab = null;
   D0100033 ds = null;
   int accessNumber = 0;
   connectorProxy = new Connector();
   try
   {
    accessNumber = connectorProxy.Login("user", "pwd", "env");
    System.out.println("Logged in successfully");
   }
   catch (CallObjectIgnoreException e)
```

```
   {
// do nothing
   }
   catch (CallObjectRetryException e)
   {
   // try one more time
    try
    {
     accessNumber = connectorProxy.Login("user", "pwd", "env");
     System.out.println("Logged in successfully");
    }
    catch (CallObjectIgnoreException ex)
    {
 // do nothing
    }
    catch (CallObjectRetryException ex)
    {
     System.out.println("EXCEPTION: :" + ex.toString());
     System.out.println("Nested Exception: "+ex.getChainedException().toString());
     System.out.println("Refer to the jasdebug.log file for more details.");
     System.exit(-1);
    }
    catch (FatalException ex)
    {
     System.out.println("Fatal Exception during login:" + ex.toString());
     System.out.println("Refer to the jas.log file for more details.");
     System.exit(-1);
    }
    catch (reject r)
    {
     System.out.println("Java Connector Exception: " + r.reason);
     System.exit(-1);
    }
   }
   catch (FatalException e)
   {
    System.out.println("Fatal Exception during login: " + e.toString());
    System.out.println("Refer to the jas.log file for more details.");
    System.exit(-1);
   }
   catch (reject r)
   {
/* This should not happen, as the Java Connector code
* now only throws one of the reject child objects.
* The documentation indicates which methods throw which
* reject child exception objects. All methods continue
* to have a signature of throws reject, however, for
* backwards compatibility (to not break existing client code).
*/
    System.out.println("Java Connector Exception: " + r.reason);
```



```
    System.exit(-1);
   }
   try
   {
    ow = (OneWorldInterface)connectorProxy.CreateBusinessObject
("Connector::OneWorldInterface", accessNumber);
    System.out.println("Got OneWorldInterface");
   }
   catch (FatalException e)
   {
    System.out.println("Fatal Exception during OneWorldInterface creation:
 " + e.toString());
    System.out.println("Refer to the jas.log file for more details.");
    System.exit(-1);
   }
   catch (reject r)
   {
    System.out.println("Java Connector Exception: " + r.reason);
    System.exit(-1);
   }
   try
   {
    ab = (AddressBook)connectorProxy.CreateBusinessObject("JDEAddressBook::
AddressBook", accessNumber);
    System.out.println("Got AddressBook");
   }
   catch (FatalException e)
   {
    System.out.println("Fatal Exception during OneWorldInterface creation:
 " + e.toString());
    System.out.println("Refer to the jas.log file for more details.");
    System.exit(-1);
   }
   catch (reject r)
   {
    System.out.println("Java Connector Exception: " + r.reason);
    System.exit(-1);
   }
   ds = ab.CreateGetEffectiveAddressParameterSet();
   ds.getmnAddressNumber().setValue("1");
   try
   {
    ds.setszCity(args[0]);
   }
   catch(StringTooLongException e)
   {
    System.out.println("Cannot set a city with length of " + args[0].length());
    System.exit(-1);
   }
   catch (reject r)
```

```
   {
    System.out.println("Java Connector Exception: " + r.reason);
    System.exit(-1);
   }
   int i=0;
   try
   {
    i = ab.GetEffectiveAddress(ds, ow, connectorProxy, accessNumber);
   }
   catch (CallObjectIgnoreException e)
   {
// do nothing
   }
   catch (CallObjectRetryException e)
   {
// try one more time
    try
    {
     i = ab.GetEffectiveAddress(ds, ow, connectorProxy, accessNumber);
    }
    catch (CallObjectIgnoreException ex)
    {
// do nothing
    }
    catch (CallObjectRetryException ex)
    {
// don't try again after second try
     System.out.println("EXCEPTION: " + ex.toString());
     System.out.println("Nested Exception: " + ex.getChainedException().
toString());
     System.out.println("Refer to the jasdebug.log file for more details.");
     System.exit(-1);
    }
    catch (FatalException ex)
    {
     System.out.println("Fatal Exception during AddressBook retrieval: " +
ex.toString());
     System.out.println("Refer to the jas.log file for more details.");
     System.exit(-1);
    }
    catch (reject r)
    {
     System.out.println("Java Connector Exception: " + r.reason);
     System.exit(-1);
    }
   }
   catch (FatalException e)
   {
    System.out.println("Fatal Exception during AddressBook retrieval: " +
e.toString());
```



```
     System.out.println("Refer to the jas.log file for more details.");
     System.exit(-1);
    }
    catch (reject r)
    {
     System.out.println("Java Connector Exception: " + r.reason);
     System.exit(-1);
    }
    String alphaname = ds.getszNamealpha();
    String address = ds.getszAddressLine1();
// get other AddressBook parameters that you want...
    if (i == 1)
    { // business function warning
     System.out.println("Warning count is " + ow.GetWarningCount());
     for (int j=0; j<ow.GetWarningCount(); j++)
     {
      System.out.println("Warning " + j + ": " + ow.GetWarningAt(j));
     }
    }
    else if (i == 2)
    { // business function error
     for (int j=0; j<ow.GetErrorCount(); j++)
     {
      System.out.println("Error " + j + ": " + ow.GetErrorAt(j));
     }
    }
    connectorProxy.Logoff(accessNumber);
   }
  }
```


CHAPTER 11

Using Java Connector Events - Guaranteed Events

This chapter provides an overview of Java connector events and discusses how to:

- Develop a Java connector events application.
- Use the Sample connector events client.

Note. This chapter is applicable only if you use guaranteed events delivery. Guaranteed event delivery is available when you use JD Edwards EnterpriseOne Tools 8.94 with JD Edwards EnterpriseOne Applications 8.11, or if you use JD Edwards EnterpriseOne Tools 8.95 and later Tools releases with JD Edwards EnterpriseOne Applications 8.9 and later Applications releases.

Refer to the Classic Events chapters if you use JD Edwards EnterpriseOne Tools 8.94 with JD Edwards EnterpriseOne Applications 8.10 or earlier releases of the JD Edwards EnterpriseOne Applications.

Understanding Java Connector Events

The Java connector provides a set of APIs that you can use to receive events when you establish a subscriber in JD Edwards EnterpriseOne with a JAVACONN transport type. When using the events portion of the Java connector, you connect directly to the JD Edwards EnterpriseOne Transaction server to receive events that have been placed in the subscriber queue.

Note. When you use the events portion of the Java connector, you do not call any business functions on the JD Edwards EnterpriseOne server. This implies that the events portion of the Java connector is not specific to the Java connector or dynamic Java connector. Therefore, the term Java connector is used throughout this chapter even though the APIs and the sample code reside in subpackages underneath the com.jdedwards.system.connector.dynamic package. All classes for the Java connector and the dynamic Java connector (not including the sample applications) reside in the Connector.jar file. Putting the Connector.jar file on the CLASSPATH is sufficient for working with either Java connector and the events operations.

Prerequisites

Whether you are developing a Java connector events application or using the sample Java connector events client, these prerequisites must exist on the machine running the events application or client sample:

- A Java Development Kit (JDK) that corresponds to the version of the JDK under which the JD Edwards EnterpriseOne Transaction server is running.

  For example, when connecting to a JD Edwards EnterpriseOne Transaction server hosted on WebSphere, you must run the Java connector events client or application using the same IBM JDK. Generally, the IBM JDK is located in <WebSphere installation directory>/java).

- An installation of IBM WebSphere MQ, if the JD Edwards EnterpriseOne Transaction Server is hosted on WebSphere.

This software comes installed as part of the installation of many different WebSphere-related software, including the WebSphere Application Client.

- A completed set of configured files for the environment:
  - jdeinterop.ini
  - jdbj.ini
  - jdelog.properties
- A JAVA_HOME environment variable that points to this JDK.
- A PATH environment variable that includes the entry, %JAVA_HOME%\bin, which assumes that JAVA_HOME has already been defined.

Additional prerequisites are required to compile and run the application or client.

- These .jar files must be on the CLASSPATH:
  - connector.jar
  - log4j.jar
  - base_JAR.jar
  - jdeNet_JAR.jar
  - system_JAR.jar

  The files can be found at <Windows client installation directory>\system\classes on the generation machine that is used for the JD Edwards EnterpriseOne environment to which you are connecting.
- These files must be copied from the Transaction server's installation directory:
  - EventProcessor_JAR.jar
  - EventProcessor_EJB.jar

  The files that you place on the CLASSPATH must be the exact same files that are on the Transaction server installation directory. The files are typically located in <Transaction Server installation>\EventProcessor\app\EventProcessor.ear.
- The JDBC driver files that correspond to the database to which you are connecting.
- The directory location for these files:
  - jdeinterop.ini
  - jdbj.ini
  - jdelog.properties

  The files must all be in the same directory. It is important to note that you put the directory in the CLASSPATH without the file names, so there is just one entry for these three files. Also, this entry must end in a slash (/), indicating that it is a directory entry and not a file name.
- If you connect to a Transaction server hosted on WebSphere, you also need these files:
  - bootstrap.jar
  - j2ee.jar
  - lmproxy.jar
  - urlprotocols.jar
  - ecutils.jar

- messagingClient.jar
- naming.jar
- namingclient.jar

The files are typically located in the <WebSphere installation directory>/lib folder. Additionally, you must put the <WebSphere installation directory>)/properties directory entry in the CLASSPATH, without an ending slash (/).

Developing a Java Connector Events Application

This section provides an overview of Java connector events application development and discusses:

- Introspection operations
- Asynchronous event sessions
- Synchronous event sessions

Understanding Java Connector Events Application Development

This list identifies the steps that you use when you write a Java class that serves as a Java connector subscriber. The steps are further explained in the code samples in this section.

- Instantiate a connector object.
- Login through the connector to the JD Edwards EnterpriseOne system.
- Instantiate an EventService object (not required for introspection operations).
- Perform introspection operations (optional).
- Create a session and receive events (optional).
- Logoff from JD Edwards EnterpriseOne.
- Shut the connector down.

You can create two types of Event Sessions, asynchronous and synchronous, to receive events through the Java connector.

Introspection Operations

The Java Connector Events API enables you to perform several introspection requests as provided in the Event IntrospectionApp.java code sample.

EventIntrospectionApp.java

This sample code shows example introspection requests:

```
import java.util.LinkedList;

import com.jdedwards.system.connector.dynamic.Connector;
import com.jdedwards.system.connector.dynamic.newevents.EventService;
```


Sample Java Connector Events Introspection application.

```
public class EventIntrospectionApp {
    public static void main(String[] args) {
          try {

            // Instantiate a Connector object
            Connector con = Connector.getInstance();

            // Login through the Connector
            int sessionID = con.login("username", "password",
"environment", "role");
```

Get the list of all events in JD Edwards EnterpriseOne. This list is returned as a LinkedList of Strings.

```
LinkedList list = EventService.getEventList(sessionID);
```

Get the template for a particular event type. This is returned as an XML template in a single String object.

```
String template = EventService.getEventTemplate(sessionID, "category",
"type", "environment");
```

Get the list of all subscriptions for the user associated with the given sessionID. This is returned as a LinkedList of com.jdedwards.pt.e1.common.events.connectorsvc.Subscription objects. This Subscription class is located in the Common_JAR.jar file.

```
LinkedList subs = EventService.getSubscriptions(sessionID);

            // Logoff the user from JD Edwards EnterpriseOne
            con.logoff(sessionID);

            // Shut the Connector down
            con.shutDown();

        } catch (Exception e) {

            e.printStackTrace();
            System.exit(-1);
        }

        System.exit(0);
 }
}
```


Asynchronous Event Sessions

With an asynchronous event session, you must create a listener class to receive events and process them according to the requirements for the event data. Once you create the listener class, you register an instance of that class with the asynchronous event session that you request. The details of these steps are listed in the MyListener.java and EventAsyncApp.java sample programs.

Additionally, the MyListener.java sample code shows that since the Asynchronous Event Session is created in CLIENT_ACKNOWLEDGE mode (illustrated in EventAsyncApp.java), the EventObject must be acknowledged to let the Transaction server know that you received the event.

MyListener.java

This sample code for the listener class not only shows the single onEvent(EventObject) method that the listener must implement, but it also shows what data you can get from the EventObject.

```
import javax.jms.IllegalStateException;

import com.jdedwards.base.datatypes.JDECalendar;
import com.jdedwards.system.connector.dynamic.SystemException;
import com.jdedwards.system.connector.dynamic.newevents.EventListener;
import com.jdedwards.system.connector.dynamic.newevents.EventObject;
```

Sample implementation of a Java Connector Asynchronous Event SessionListener.

```
public class MyListener implements EventListener {
```

Permits the listener to receive an event when it has been delivered from the Transaction Server.

@param event the event

```
public void onEvent(EventObject event) {
```

Do some processing here with the event that is sent by the Transaction Server. The onEvent(EventObject) method is called once for every event that is delivered.

*The event category: "RTE", "XAPI", or "ZFILE".

```
String category = event.getCategory();
```

The event type, such as "RTSOOUT".

```
String type     = event.getType();
```

The JD Edwards EnterpriseOne environment in which the event was generated.

```
String environment = event.getEnvironment();
```

The global sequence number of the event.

```
long sequenceNumber = event.getSequenceNumber();
```

The date and time stamp of the event.

```
JDECalendar date = event.getDateTime();
```

The XML content of the event as a single String object.*/

```
String xmlPayload = event.getXMLPayload();
```

If you created an EventSession with CLIENT_ACKNOWLEDGE mode, you must acknowledge each message you receive. Otherwise the event will be redelivered according to the Transaction Server JMS Provider's logic.

```
            try {

            event.acknowledge();

          } catch (IllegalStateException e) {
```

This Exception will be thrown if the session associated with this event has already been closed.

```
       } catch (SystemException e) {
```

This Exception will be thrown if the original event could not be acknowledged (duplicate event delivery is likely in this scenario).

```
         }
      }
}
```

EventAsyncApp.java

The asynchronous-specific calls in this asynchronous event application (AsyncEventApp.java) are illustrated in this code sample. Between the eventSession.start and the eventSession.stop method calls, you would normally solicit user input or wait for some type of intervention to let the class know that event delivery needs to stop.

```
import com.jdedwards.system.connector.dynamic.Connector;
import com.jdedwards.system.connector.dynamic.newevents.AsyncEventSession;
import com.jdedwards.system.connector.dynamic.newevents.EventService;
import com.jdedwards.system.connector.dynamic.newevents.EventSession;
```

Sample Java Connector Asynchronous Event application

```
public class EventAsyncApp {

    public static void main(String[] args) {

  try {
```

Instantiate a Connector object

```
            Connector con = Connector.getInstance();
```


Login through the Connector to JD Edwards EnterpriseOne

```
            int sessionID = con.login("username", "password",
   "environment", "role");
```

Instantiate an EventService object

```
             EventService service = EventService.getInstance();
```

Create a synchronous event session in CLIENT_ACKNOWLEDGE mode.

```
            AsyncEventSession eventSession = service.getAsyncEventSession
   (sessionID, EventSession.CLIENT_ACKNOWLEDGE);
```

Register a listener object which you have created

```
             eventSession.registerListener(new MyListener());
```

Start the delivery of events to the listener

```
            eventSession.start();
```

Stop the delivery of events to the listener. Note that you can continuously alternate between calls to start() and stop() as long as you do not call the close() method.

```
            eventSession.stop();
```

Close the event session. No other operations on the event session are possible at this point.

```
             eventSession.close();
```

Logoff the user from JD Edwards EnterpriseOne

```
            con.logoff(sessionID);
```

Shut the Connector down

```
            con.shutDown();

          } catch (Exception e) {

              e.printStackTrace();
              System.exit(-1);

          }

          System.exit(0);
      }
   }
```

Synchronous Event Sessions

With synchronous event sessions, you receive only one event at a time. No listener class is involved with this type of session.

EventSyncApp.java

The three ways to receive an event, along with an explanation of functionality, are illustrated in this EventSyncApp.java class sample code. This sample code uses the AUTO_ACKNOWLEDGE acknowledgement mode:

```
import com.jdedwards.system.connector.dynamic.Connector;
import com.jdedwards.system.connector.dynamic.newevents.EventObject;
import com.jdedwards.system.connector.dynamic.newevents.EventService;
import com.jdedwards.system.connector.dynamic.newevents.EventSession;
import com.jdedwards.system.connector.dynamic.newevents.SyncEventSession;
```

Sample Java Connector Synchronous Events application.

```
public class EventSyncApp {

    public static void main(String[] args) {

  try {
```

Instantiate a Connector object

```
Connector con = Connector.getInstance();
```

Login from the Connector to JD Edwards EnterpriseOne

```
int sessionID = con.login("username", "password",
 "environment", "role");
```

Instantiate an EventService object

```
EventService service = EventService.getInstance();
```

Create a synchronous event session in AUTO_ACKNOWLEDGE mode

```
 SyncEventSession eventSession =
 service.getSyncEventSession(sessionID,
EventSession.AUTO_ACKNOWLEDGE);
```

Start the delivery of events

```
eventSession.start();
```

The receive() method will not return control to the caller until an event is delivered.

```
EventObject event1 = eventSession.receive();
```

Do some processing of the event data here. Refer to the sample class (MyListener.java) for a list of the methods that can be called on the EventObject class.

The receive(long timeout) method will return control to the caller if the timeout value (in milliseconds) elapses without an event being delivered. Of course, if an event is delivered before the timeout value elapses, the EventObject will be returned to the caller.

```
EventObject event2 = eventSession.receive(5000);
```


Do some processing of the event data here. Refer to the sample 'MyListener.java' class for a list of the methods that can be called on the EventObject class.

The receiveNoWait() method either immediately returns an EventObject to the caller if an event is waiting to be delivered or returns null if no event is waiting.

```
EventObject event3 = eventSession.receiveNoWait();
```

Do some processing of the event data here. Refer to the sample 'MyListener.java' class for a list of the methods that can be called on the EventObjectclass.

Stop the delivery of events. Note that you can continuously alternate between calls to start() and stop() as long as you do not call the close() method.

```
eventSession.stop();
```

Close the event session. No other operations on the event session are possible at this point.

```
eventSession.close();
```

Logoff the user from JD Edwards EnterpriseOne

```
con.logoff(sessionID);
```

Shut the Connector down

```
con.shutDown();

} catch (Exception e) {

    e.printStackTrace();
    System.exit(-1);

}

System.exit(0);
    }
}
```

Using the Sample Connector Events Client

This section provides an overview of connector events client tool and discusses:

1. Using the Connector Events Client tool.
2. Configuring the sample connector events client.
3. Running the sample connector events client.
4. Resolving Connector Events Client tool issues.

Understanding Connector Events Client Tool

The connector events client is a Java-based graphical tool that enables you to log in to JD Edwards EnterpriseOne and receive events that you have subscribed to from the JD Edwards EnterpriseOne Transaction server. This tool enables all possible event operations, including all of the introspection requests as well as the creation of both asynchronous and synchronous event sessions.

Prerequisites for Using the Sample Connector Events Client

In addition to meeting the requirements listed in the Prerequisites for Understanding Java Connector Events section, you must also verify:

- The Transaction server is running.
- The user ID that you use to log in to the tool is a user ID that is an active subscriber with at least one active subscription.
- A Java Runtime Environment (JRE) version 1.4 or later is installed on your machine.

  You can download a valid JRE from Sun Developer Network web site.

See Also

Chapter 11, “Using Java Connector Events - Guaranteed Events,” Prerequisites, page 135

Sun Developer Network (SDN), http://java.sun.com/j2se/downloads/index.html

Using the Connector Events Client Tool

You sign in to the connector events client tool through the login window. Once you have successfully signed in, you can perform any of the introspection operations without creating an event session. All error messages are displayed in the bottom pane. If you receive an error message that is not explained sufficiently, you can look in the debug log file of the tool to obtain more information.

The buttons that enable you to create a new event session prohibit you from entering an invalid sequence or combination (such as starting event delivery without opening a session). Once you start receiving events, the event sequence numbers for received events appear in the Event List window. If you select an event sequence number, the event details for that event appear in the Event Data window. Additionally, the XML content for all received events is automatically created as an XML file in the tool’s log directory, regardless of whether you select the sequence number for the event.

To use the tool, you must build, configure, and then run the tool. The tool is shipped to you as source code so that you can inspect the usage of the connector events APIs. You can find the entire source code in a single jar file: connector_samples_src.jar. This file should be located in the <Windows client generation machine installation directory>/system/classes/samples folder.

Configuring the Sample Connector Events Client

This section provides steps for configuring the sample connector events client.

To configure the Sample Connector Events Client

Use these steps to configure the sample connector events client:

1. Create a C:\ConnectorEventsClient directory.

   If a directory with this name already exists, rename the existing directory before you create a new directory.

2. Unzip the Connector Events Client.zip file to the newly created directory on your C drive.

   Make sure to unzip the file with the full path information for each file in the Zip file.

3. Configure the files in your C:\ConnectorEventsClient\config directory.

   Make sure that the configured files have the .templ file extension removed from them. The proper file names for this directory are:

   - jdbj.ini
   - jdeinterop.ini
   - jdelog.properties

   Configure your jdbj.ini and jdelog.properties files according to your environment. See your JD Edwards EnterpriseOne systems administrator if you do not know the appropriate values for these files. You should name your jdbj.ini file with the same file name that is configured on your Transaction server.

   Configure your jdeinterop.ini file with these values:

Section	Setting	Value
[EVENTS]	eventServiceURL	http://machine_name:port/e1events/EventClientService The machine name is the name of your WebSphere Transaction Server and the port is the port for your WebSphere Transaction Server.
[SECURITY]	SecurityServer	Name of your JD Edwards EnterpriseOne Security Server.
[JDENET]	serviceNameConnect	The port you are connecting to on your JD Edwards EnterpriseOne Security Server.

4. Add the appropriate JDBC driver files to your C:\ConnectorEventsClient\lib directory.

   See your JD Edwards EnterpriseOne systems administrator to determine which driver file to use.

5. Edit the C:\ConnectorEventsClient\setDynConNewEventDriver.bat file, change it to point to the location of your installed JRE.

Running the Sample Connector Events Client

Use these steps to run the sample Connector Events Client:

1. Navigate to the C:\ConnectorEventsClient directory.
2. Double-click the runDynConNewEventDriver.bat file.
3. On the Java Connector EnterpriseOne signon window, enter your JD Edwards EnterpriseOne credentials, and then select the OK button.
4. Click Open Session and then click Start to receive events for which you have subscribed.

The event numbers for any events that are waiting for you should appear in the Event List window. If you select an event number, the event data for the selected event appears in the Event Data window. The XML content for each event is also placed in your C:\ConnectorEventsClient\logs directory.

Resolving Java Connector Events Client Tool Issues

This table discusses potential problems that you might encounter when using the Java Connector Events Client tool, along with possible solutions.

Problem	Possible Solution
I can't get past the sign-on screen.	Try entering all of your credentials (username, password, environment, and role) in all capital letters.
My C\ConnectorEventsClient\logs directory is full, and I would like to delete some of the .log and .xml files.	You may delete any files that this directory at any time. However, if your Connector Events Client application is running, some of the files might be locked.
Why are there orbtrc...txt files in my C:\ConnectorEventsClient directory?	These files are created by WebSphere runtime code. You may delete these files at any time. However, if your Connector Events Client application is running, some of these files might be locked.
An error message that I don't understand appears in the Error Messages window.	Look in your C:\ConnectorEventsClient\logs directory for the jasdebug_date.log file that corresponds to the appropriate date. Often a more explanatory error message can be found in this file.
I clicked the ReceiveAndWait button, and now the interface is frozen.	This happens when you click the ReceiveAndWait button and there is no event waiting for you on the Transaction Server. ReceiveAndWait means that you are willing to wait indefinitely for an event to be generated and delivered to you. The interface freezes in this instance until an event is delivered. If you are not willing to wait, click the ReceiveNoWait button.

CHAPTER 12

Understanding J2EE Connector Architecture Resource Adapter

This chapter discusses:

- J2EE Connector Architecture Resource Adapter.
- JCA 1.0 Specification optional features.
- Assembly and components.
- Deployment and configuration.
- Common client interface.
- Signon types.
- Subclasses.
- Input and output data.
- Logs.
- Exceptions.
- Samples.
- Checklist for resolving issues.

J2EE Connector Architecture Resource Adapter

The JD Edwards EnterpriseOne J2EE Connector Architecture (JCA) resource adapter enables Java2 Platform, Enterprise Edition (J2EE) components to use a standard interface to connect to the JD Edwards EnterpriseOne system. A resource adapter is a system-level software driver that enables J2EE components to communicate with a back-end enterprise information system (EIS) through a JCA-compliant application server when a resource adapter for the specific EIS is deployed to the server. J2EE components consist of Servlets, JavaServer Pages (JSPs), and Enterprise JavaBeans (EJBs).

J2EE components and applications built with J2EE components can execute business functions through the JD Edwards EnterpriseOne JCA resource adapter. JD Edwards EnterpriseOne business functions are accessed through the JCA standard client interface, the Common Client Interface (CCI). The JD Edwards EnterpriseOne JCA resource adapter is fully compliant to the Java2 Platform, Enterprise Edition (J2EE) JCA 1.0 Specification and should work with any application server that is J2EE 1.4 certified.

Note. Some application servers are known to not be J2EE 1.4 certified but they support some J2EE 1.4 features, including JCA 1.0. Check with the application server vendor to determine whether the application server supports JCA1.0.

See Also

J2EE Connector Architecture, http://java.sun.com/j2ee/connector/

JCA 1.0 Specification Optional Features

The JCA 1.0 Specification identifies optional features for developing a resource adapter. This table addresses the level of support that the JD Edwards EnterpriseOne JCA resource adapter provides for the optional features identified in the JCA 1.0 Specification.

Feature	Level of Support
Transactions	The JD Edwards EnterpriseOne JCA resource adapter is classified as an XA Transaction resource adapter. The JD Edwards EnterpriseOne JCA resource adapter permits either no transactions during business function calls, transactions local to JD Edwards EnterpriseOne during those same calls (local transaction), and one-phase commit (1PC) XA transactions (transactions that span multiple enterprise information systems).
Client Interface	The JD Edwards EnterpriseOne JCA resource adapter supports the optional common client interface (CCI), which is modeled after the Java Database Connectivity (JDBC) client API. This relatively simple Java API should significantly reduce the learning curve for using the JD Edwards EnterpriseOne JCA resource adapter.
Reauthentication	The JD Edwards EnterpriseOne JCA resource adapter does not support the switching of a set of JD Edwards EnterpriseOne user credentials on an existing JD Edwards EnterpriseOne user session. User credentials are usually a concern of the application server and should not affect client development.
Input/Output Records	The JD Edwards EnterpriseOne JCA resource adapter supports the MappedRecord interface, which is a data type of key-value pairs. The MappedRecord interface is further discussed in the Input/Output Data section of this document. The CCI interfaces IndexedRecord and ResultSet are not supported as they are not relevant to the type of output from business functions.
Authentication	The JD Edwards EnterpriseOne JCA resource adapter supports BasicPassword authentication, which indicates to the application server how to handle container-managed signon. The resource adapter does not support any other form of authentication, such as Kerberos authentication through the GenericCredential interface. The Signon Types section of this document provides more information about authentication with the resource adapter.

Feature	Level of Support
ManagedConnectionFactory Properties	The JCA Specification identifies these properties as standard; however, these properties are optional properties for the ManagedConnectionFactory class, which is the main class configured with JD Edwards EnterpriseOne specific properties during deployment of the resource adapter: • ServerName • PortNumber • UserName • Password • ConnectionURL The JD Edwards EnterpriseOne JCA resource adapter supports the UserName and Password properties, as the other properties are either irrelevant properties or are configured elsewhere in the resource adapter. The Deployment Settings section of this document addresses other properties that are defined by the JD Edwards EnterpriseOne JCA resource adapter. **Note.** The deployment tool of the particular J2EE application server might list these properties as configurable for the resource adapter. The JD Edwards EnterpriseOne JCA resource adapter does not use values that you assign to these properties (other than that for UserName and Password).
Number of Deployed Resource Adapters	The JCA Specification allows for the possibility of deploying the same resource adapter multiple times on a given application server. This provides for potential connectivity to multiple versions of the same EIS for a one resource adapter-to-many-EIS version ratio. The JD Edwards EnterpriseOne JCA resource adapter supports the deployment of only one JD Edwards EnterpriseOne JCA resource adapter per application server (essentially one resource adapter per virtual machine). **Note.** You can install different JCA resource adapters (those other than for JD Edwards EnterpriseOne) on the same application server.
Non-Managed Scenario	The JD Edwards EnterpriseOne JCA resource adapter must be used with an application server or an application client. If you want to access JD Edwards EnterpriseOne business functions through Java outside of an application server or application client, you should use the Java connector directly.

See Also

JCA Java documentation (APIs), http://java.sun.com/j2ee/apidocs-1_0-fr/api/index.html

Assembly and Components

The packaging of a resource adapter is defined in the JCA 1.0 Specification. However, because some application servers require additions to the standard Resource Adapter Archive (RAR) file, it is not possible to distribute a single RAR file that can be deployed to all application servers. Consult the application server documentation for instructions on how to use the assembly tool and to understand what additional components might be required for a resource adapter to be operational with the application server. Typically, an additional deployment descriptor is required. Additional information required by an application server is usually for performance tuning and for configuration settings.

Components

A RAR file is a file that is in Java Archive (JAR) File Format with a .rar extension instead of a .jar extension. The file structure for a RAR file is:

- /META-INF/ra.xml
- /<all necessary JAR files>

The ra.xml file is the standard resource adapter deployment descriptor and must be put in the META-INF directory of the RAR file. The ra.xml file must be named *exactly* ra.xml. The ra.xml file for the JD Edwards EnterpriseOne JCA resource adapter is provided in the system/classes/samples directory on the JD Edwards EnterpriseOne CD. All other JAR files go in the root directory of the RAR file. The JAR files are provided in the system/classes directory on the same CD. The required resource adapter JAR files include:

- owra.jar.
- connector.jar.
- log4j.jar.
- base_JAR.jar.jar.
- jdeNet_JAR.jar.
- system_JAR.jar.
- xerces.jar.
- JDBC driver .jar files supplied by the database vendor.

Note. Only use the versions of these JAR files that come with the JD Edwards EnterpriseOne distribution.

When the RAR file is finally created, the META-INF directory of the RAR file might contain a Manifest.mf file. The Java JAR tool usually creates the Manifest.mf file automatically. The Manifest.mf file complies with the JAR file format, and it is acceptable for the Manifest.mf file to be in the RAR file.

Deployment and Configuration

The methods and tools for configuring and deploying a resource adapter vary between application servers and even between versions of the same application server. Consult the application server documentation for information about how to configure and deploy a resource adapter. Two separate methods exist for deploying a resource adapter. The first method is deploying the resource adapter as a standalone resource adapter. This permits all applications deployed on the application server to access the same resource adapter. The second method involves packaging the resource adapter within an enterprise application (EAR) file. This permits only those components in the EAR file to have access to the resource adapter. The sample applications provided with the JD Edwards EnterpriseOne JCA resource adapter use the second method.

Additional settings required for the JD Edwards EnterpriseOne JCA resource adapter to be deployed and to operate correctly include:

- Security permissions.
- jdeinterop.ini settings.
- jdbj.ini settings.
- jdelog.properties settings.
- CLASSPATH settings.
- Configurable properties.
- Java naming directory interface settings.

Note. Only one JD Edwards EnterpriseOne JCA resource adapter can be deployed in standalone mode per application server.

Security Permissions

The JCA 1.0 Specification defines the standard Java security permissions that must be granted to all resource adapters by an application server. The JD Edwards EnterpriseOne JCA resource adapter needs additional security permissions to operate. These permissions are listed in the deployment descriptor (ra.xml file). Most application servers dynamically grant these permissions to the resource adapter during deployment. Some application servers have other methods of granting the resource adapter additional permissions, including modifying a Java security policy file, which might require that you restart the application server to take effect.

If the application server does not dynamically grant the security permissions to a resource adapter based on the contents of the deployment descriptor, you need to grant the resource adapter the permissions listed in the security-permission-spec elements of the deployment descriptor. If the application server throws a SecurityException while running an application associated with the JD Edwards EnterpriseOne JCA Resource Adapter, it is possible that the necessary security permissions are not being granted to the resource adapter.

jdeinterop.ini Settings

Because the resource adapter is built on top of the Java connector, it is necessary to configure the appropriate settings in the jdeinterop.ini file to make the Java connector operational. The resource adapter introduces no new settings into the jdeinterop.ini file.

jdbj.ini Settings

You must set up the jdbj.ini file.

See *JD Edwards EnterpriseOne Tools 8.96 HTML Web Server Installation,* Appendix D, Parameters and Values for the jdbj.ini File

jdelog.properties Settings

The JCA 1.0 Specification permits resource adapter-specific logging messages to be sent to a separate log file, which can be configured according to the application server (see the application server documentation). The messages that are sent to this log file are redundant to and are a subset of the messages that are sent to the log file defined in the jdelog.properties file. This redundancy is an intentional JD Edwards EnterpriseOne JCA resource adapter design decision for this reason:

The JCA logging mechanism does not provide a method for logging messages from the connector on which the resource adapter is built. The logging properties file permits all logging messages from the connector as well as the resource adapter to be logged in a central location.

CLASSPATH Settings

The JD Edwards EnterpriseOne JCA resource adapter requires that the complete path to the jdelog.properties file be placed in the server's CLASSPATH. This path cannot include the name of the file, and the path must end with a slash, which designates that the last item in the path is a directory and not a file. The name of the properties file is required to be *jdelog.properties*. The logging mechanism looks for the logging properties file in all directories in the CLASSPATH.

The JDBC driver for the JD Edwards EnterpriseOne database must be in the server's CLASSPATH so that the proper database connections can be made.

Note. Some servers require all of the JAR files within the resource adapter RAR file to be placed in the server's CLASSPATH. If you encounter a *NoClassDefFoundError* while running a Web application that is using the resource adapter, try putting all of these JAR files in the server's CLASSPATH and restarting the server. Consult the server documentation for further ClassLoader issues.

Configurable Properties

The JD Edwards EnterpriseOne JCA resource adapter deployment descriptor (ra.xml file) contains properties that must be assigned values specific to the environment. This table identifies the configurable properties and describes the information required.

Property	Required Information
owVersion	The version of JD Edwards EnterpriseOne to which the resource adapter connects. This property is for display purposes only and can contain any value. The value you enter in this property is not validated against the JD Edwards EnterpriseOne installation.
username	Use this property for a JD Edwards EnterpriseOne user when neither the container nor the application supplies a set of JD Edwards EnterpriseOne user credentials.
password	Use this property for a JD Edwards EnterpriseOne user when neither the container nor the application supplies a set of JD Edwards EnterpriseOne user credentials.

Property	Required Information
environment	It is possible in a resource adapter web application to map a user's web credentials to a set of JD Edwards EnterpriseOne user credentials. This mapping, which is called container-managed signon, prevents the user from having to present different credentials multiple times while using a single web application. Container-managed signon maps a given user name and password to a JD Edwards EnterpriseOne user name and password. Container-managed sign-on mapping is specific to each application server. For JD Edwards EnterpriseOne, the environment property is used to add a valid JD Edwards EnterpriseOne environment to the user name and password mapped by the application server, which permits proper JD Edwards EnterpriseOne signon. If you use container-managed signon, you must assign a value to this property.
role	In addition to user name, password, and environment, JD Edwards EnterpriseOne signon requires a role. The role property has a default value of **ALL*, which enables the user to assume all valid roles for the JD Edwards EnterpriseOne user name. You do not need to assign a value for role if this is the value you want to use.

Consult the documentation for the application server to determine if other deployment settings are required.

Java Naming and Directory Interface Settings

For communication between the web application and the JD Edwards EnterpriseOne JCA resource adapter, the web application must perform a Java Naming and Directory Interface (JNDI) lookup of the ConnectionFactory of the resource adapter. You are allowed to configure multiple ConnectionFactory instances for each resource adapter. This permits setting different values for the configurable properties listed in the previous section. The web application obtains a JD Edwards EnterpriseOne connection and interacts with JD Edwards EnterpriseOne through the ConnectionFactory. The method of assigning a JNDI name to the ConnectionFactory for the JD Edwards EnterpriseOne JCA resource adapter is specific to and documented by the application server.

When you add a ConnectionFactory through the application server, you are provided with a method for assigning values to the configurable properties for each ConnectionFactory.

Common Client Interface

The Common Client Interface (CCI) is the JCA-recommended client API for all resource adapters. The JD Edwards EnterpriseOne JCA resource adapter provides an implementation of CCI as the client interface.

Implementing the Common Client Interface

This example code shows how to implement a CCI for the JD Edwards EnterpriseOne JCA resource adapter. In the example code, the elements in quotes have descriptive names and must have values valid to the environment in a real Java class. The line numbers in the example code are not part of the code but are for reference in subsequent paragraphs.

```
import com.jdedwards.system.connector..dynamic.jcaplugin.
ImageBSFNInteractionSpecImpl;
import com.jdedwards.system.jca.cci.ConnectionSpecImpl;
```

```
import javax.naming.Context;
import javax.naming.InitialContext;
import javax.naming.NamingException;
import javax.resource.ResourceException;
import javax.resource.cci.Connection;
import javax.resource.cci.ConnectionFactory;
import javax.resource.cci.Interaction;
import javax.resource.cci.MappedRecord;
import javax.resource.cci.RecordFactory;
import javax.resource.cci.ResourceWarning;

public class SomeClass {

public void someMethod() {

 try {
  // get the naming context
  Context nc = new InitialContext();

  // lookup the connection factory
  ConnectionFactory conFact = (ConnectionFactory)nc.lookup("Resource
Adapter JNDI Name");

  //1. create a ConnectionSpec
ConnectionSpecImpl conSpec = new ConnectionSpecImpl("username",
"password", "environment", "role");

  //2. get the Connection to JD Edwards EnterpriseOne
Connection con = conFact.getConnection(conSpec);

  // create an Interaction
  Interaction ix = con.createInteraction();

  // create and populate the InteractionSpec
  OWBSFNInteractionSpecImpl ixSpec = new OWBSFNInteractionSpecImpl();
  ixSpec.setBusinessFunction("Business Function Name");

  // get a RecordFactory
  RecordFactory rf = conFact.getRecordFactory();

  //3. create the input MappedRecord
MappedRecord inputRecord = rf.createMappedRecord("any descriptive
name");

  //4. populate the input MappedRecord with the input values
inputRecord.put("Business Function Parameter Name",
         " Business Function Parameter Value");

  //5. execute the Business Function, putting the results in the output
  //   MappedRecord
```



```
MappedRecord outputRecord = (MappedRecord)ix.execute(ixSpec, inputRecord);

  // get results
  Object value = outputRecord.get("Business Function Parameter Name");

  // get Business Function warnings, if any
  ResourceWarning warning = ix.getWarnings();

  // close the Interaction
  ix.close();

  // close the Connection
  con.close();
 } catch (ResourceException e) {
  // handle resource adapter-related Exceptions here
 } catch (NamingException e) {
  // handle JNDI-related Exceptions here
 }
}
}
```

Signon Types

The JD Edwards EnterpriseOne JCA resource adapter provides these types of JD Edwards EnterpriseOne signons:

- Container-managed signon
- Component-managed signon

Container-Managed Signon

When container-managed signon is used, the application server maps a web application user to a given JD Edwards EnterpriseOne user. In this case, JD Edwards EnterpriseOne user credentials are not provided in the CCI code. If you use container-managed signon, line 1 of the example code would not exist, as you do not need to create an instance of the ConnectionSpecImpl class. Line 2 of the example code would be changed to this:

```
Connection con = conFact.getConnection();
```

Component-Managed Signon

When component-managed signon is used, the code provides specific JD Edwards EnterpriseOne credentials (either through coding specific credentials or by obtaining JD Edwards EnterpriseOne credentials through user entry in the web application) to the JD Edwards EnterpriseOne JCA resource adapter for JD Edwards EnterpriseOne signon. In the example code, lines 1 and 2 illustrate component-managed signon. In line 1 of the example code, an instance of the ConnectionSpecImpl class is first created with the JD Edwards EnterpriseOne user credentials. That instance is then passed to the getConnection method.

Component-managed signon is also known as application-managed signon.

Subclasses

The import statements at the top of the example code illustrate that most of the classes that you use to interact with the JD Edwards EnterpriseOne JCA resource adapter are JCA classes (those classes in the javax.resource package and sub-packages) and not JD Edwards EnterpriseOne-specific implementations of JCA interfaces. JD Edwards EnterpriseOne software provides these implementation classes:

- ConnectionSpecImpl
- xxxxInteractionSpecImpl

The ConnectionSpecImpl class supplies the required JD Edwards EnterpriseOne user credentials to the getConnection method. The ConnectionSpecImpl class is one of the signon types. Line 1 in the example code shows how to use the ConnectionSpecImpl class.

The purpose of the xxxxInteractionspecImpl class is to establish the necessary business function information before execution in JD Edwards EnterpriseOne. The xxxxInteractionSpecImpl classes vary, depending on the type of business function spec source. The business function spec source is a file or location that describes a business function. Each implementation class, which is a concrete class of the javax.resource.cci.InteractionSpec interface, includes methods that set values. These setter methods must be called and given values before executing the business function through the resource adapter.

ImageBSFNInteractionSpecImpl

The ImageBSFNInteractionSpecImpl implementation class gets the business function spec from an XML image file, which must be generated by the dynamic Java connector beforehand.

Class: com.jdedwards.system.connector.dynamic.jcaplugin.ImageBSFNInteractionSpecImpl

Method: setBusinessFunction(String value)

Sets the *exact* name of the business function.

Method: setImageFilename(String value)

Sets the complete path and filename of the dynamic Java connector JD Edwards EnterpriseOne spec image that contains the definition of the corresponding business function.

OWBSFNInteractionSpecImpl

The OWBSFNInteractionSpecImpl implementation class gets the business function spec directly from a call to JD Edwards EnterpriseOne. This method might take a little longer to execute a business function the first time the business function is called. The business function is stored in memory, and execution should be quicker in subsequent calls.

Class: com.jdedwards.system.connector.dynamic.jcaplugin.OWBSFNInteractionSpecImpl

Method: setBusinessFunction(String value)

Sets the *exact* name of the business function.

Input and Output Data

The MappedRecordImpl class handles both sending input data to the resource adapter and receiving the output data that is the result of executing the business function. Lines 3 and 4 of the example code illustrate inputting data, and line 5 illustrates obtaining the output data. A MappedRecord is a correlation of key/value pairs. The key represents the exact business function parameter name, and value defines the key.

Input data for values can be supplied in one of these ways:

- Use a string.
- Use a native Java data type.

The JD Edwards EnterpriseOne JCA resource adapter examines the input data on a parameter-by-parameter basis. If the input data type is string, the resource adapter attempts to convert the input data to the appropriate Java data type for the specified parameter. If both the actual parameter type and the input data are string, the resource adapter passes the input data through unchanged. If the input parameter is a native Java data type, the resource adapter passes the input data through unchanged.

If the native Java data type is incorrect or if the parameter name is invalid for the given business function, the resource adapter throws an exception.

This table lists the business function types and their corresponding native Java data type:

JD Edwards EnterpriseOne Data Type	Native Java Data Type
ID	java.lang.Integer
char (length of only 1)	java.lang.Character
JDEDATE	java.util.Date
Calendar	com.jdedwards.base.datatypes.JDECalendar
MATH_NUMERIC	com.jdedwards.system.lib.MathNumericImpl
char (variable length)	java.lang.String

The output of all business functions result in the data in the MappedRecordImpl being in the native Java data types. If you prefer only string-formatted output, you can make this call on the output MappedRecordImpl for each parameter retrieved:

String value = outputRecord.get("parameter name").toString();

Logs

Message logging for the JD Edwards EnterpriseOne JCA resource adapter is controlled by the jdelog.properties file.

Exceptions

The parent Exception class for all exceptions thrown by the JD Edwards EnterpriseOne JCA resource adapter is javax.resource.ResourceException.

See Also

JCA Javadoc, http://java.sun.com/j2ee/apidocs-1_0-fr/api/index.html

Samples

The samples supplied with the resource adapter illustrate how to use the resource adapter's API, as well as the JCA API, and how to demonstrate the functionality of the resource adapter. Address Book Query, Sales Order Entry, and Purchase Order Entry are included samples. The source code along with the compiled classes are delivered on the JD Edwards EnterpriseOne Java Server CD in the system/classes/samples directory in the JCASamples.ear file and the JCASamples_WebSphere.ear file, which includes WebSphere 5.x-specific deployment files.

The sample applications consist of a group of servlets, which provide the HTML for the display of the samples, and a group of stateful session Enterprise JavaBeans (EJBs) that access the JD Edwards EnterpriseOne JCA resource adapter. The resource adapter is bundled inside the .ear files and is only available to the sample applications when deployed to the application server.

Prepare the Samples for Deployment

These customizations must be performed to the .ear file before it can function correctly.

- JDBC driver .jar file.
- Configuration files.
- Samples for the application server.

JDBC Driver .jar File

The JDBC driver .jar file supplied by the JD Edwards EnterpriseOne database vendor must be packaged inside the .ear file. Since the .ear file is in a Zip format, you can use a Zip program to add the necessary files. Place the JDBC driver .jar files in the root directory of the .ear file (no path for those files). The CLASSPATH in the manifest.mf file on the .ear file includes the expected filenames for the JDBC .jar files for three database vendors without actually being included in the driver files themselves:

- SQL Server: msbase.jar, msutil.jar, mssqlserver.jar
- Oracle: classes 12.jar
- DB2: jt400.jar

If the file names of the JDBC driver .jar files are different, add those file names to the manifest.mf file that is located inside the meta-inf directory of the JCA Samples RAR.rar file within the sample application.ear file you are using. Be sure to preserve the meta-inf path for the manifest.mf file when you add it back into the file.

Configuration Files

You must configure these files:

- jdbj.ini
- jdeinterop.ini
- jdelog.properties.

These configuration files are in the config directory of the sample application EAR file. After you customize the settings, be sure to place the files back into the EAR file in the config directory.

Samples for the Application Server

A generic JCASamples.ear file and a WebSphere 5.x-specific JCASamples_websphere.ear file are provided. The application server might need additional information for some of the components contained in the EAR file. This is a list of the sample components:

- JCASamplesEJB.jar

  A JAR file that contains the Enterprise JavaBean (EJB) classes used by the samples.

- JCASamplesRAR.rar

  A rar file that contains only the resource adapter deployment descriptor. The dependent JAR files for the resource adapter are contained in the parent directory of the EAR file, as they need to be used by the entire application.

- JCASamplesWeb.war

  A WAR file containing the servlets for the sample applications.

If you use the generic JCASamples.ear file to deeply the sample applications to the application server, and they do not operate correctly, you might need to unpack each of the files individually (.ear, .jar, .rar, and .war files) and repack them with the application server's assembly tool. This step usually enables the tool to place new files and information in existing files that enable the application to operate correctly for that application server.

Deploy the Sample Applications

These general steps must be completed for deploying the sample applications to any application server:

- Start the application server.
- Start the administrative console (whatever application that ships with the application server that enables you to deploy applications).
- Install the enterprise application.
- Add a connection factory for the resource adapter with a JNDI name of OneWorldJCAAdapter (with that exact spelling).
- Restart the application server.

The application server may require additional steps not listed here (see the application server documentation for deploying enterprise applications).

Deploy the Sample Applications to WebSphere 5.x

Use these steps to deploy the sample applications on WebSphere 5.x:

1. Start WebSphere.
2. Start the WebSphere Administrative Console.
3. Log on to the WebSphere Application Server Administrative Console using any ID.

4. On the WebSphere Administrative Console, expand the applications node on the left side of the screen, and then click the Install New Applications link.

   Preparing for the Application Installation appears on the right side of the screen.

5. In the Preparing for the application installation portion of the screen, click Browse, and then select JCASamples_WebSphere.ear file.

6. Click Next.

   Continue to click Next on all successive screens until the final Summary screen presents a Finish button at the bottom of the screen. Accept the default values provided on each of the screens without altering any of them.

7. On the final Summary screen, click Finish.

   WebSphere automatically generates the necessary EJB deployment code.

8. At the bottom of the screen, after the notice that the Application JCA samples installed successfully, click the Save to Master Configuration link.

   A Message box (indicating that changes have been made to the local configuration) and an Enterprise Application Save section that includes a Save to Master Configuration box appear.

9. In the Save to Master Configuration box, click the Save button.

   You are returned to the main screen.

10. On the left side of the main screen, click the Enterprise Applications link from the menu.

    A list of the installed applications appears on the right side of the screen.

11. On the right side of the screen, click JCASamples from the list that appears under Enterprise Applications.

12. In the Related Items area at the bottom of the next screen, click the Connector Modules link.

13. On Connector Modules, click JCASamplesRAR.rar.

14. On the screen with a Configuration tab, scroll to the Additional Properties area and click the Resource Adapter link.

15. On the next screen with a Configuration tab, scroll to the Additional Properties area, and then click the J2C Connection Factories link.

16. On the J2C Connection Factories screen, click the New button to establish a new J2C Connection Factory.

17. Under the Configuration tab on the New screen, enter any value for the Name field and OneWorldJCAAdapter for the JNDI name.

    The value you enter for the Name field is used for display purposes only.

18. Scroll to the bottom of the screen, and then click the OK button.

19. In the Message box at the top of J2C Connection Factories screen, click the Save link.

20. In the Save to Master Configuration area on the Save screen, click the Save button.

21. From the menu bar (at the top of the screen), click Logout.

22. Stop and restart the sever to make the application is available to run.

Run the Sample Applications

After you configure and deploy the sample applications, you can run each of the sample applications, provided that the JD Edwards EnterpriseOne Server you are accessing is operational. Use these URLs to access the samples:

- AddressBook Query: http://<app http://<app server name>|<app server port>/JCASamplesWeb/ABLogin
- SalesOrder Entry: http://<app server name>|<app server port>/JCASamples Web/SOLogin
- PurchaseOrder Entry: http://<app server name>|<app server port>/JCASamplesWeb/POLogin

Checklist for Resolving Issues

If the system is not working, use this checklist to ensure you have the proper setup:

- The directory location of the jdelog.properties file must be in the server's CLASSPATH.

  For example, if the jdelog.properties file is in this location:

  C:\JCA\logs\jdelog.properties

  you must have this entry in the server's CLASSPATH:

  C:/JCA/logs/

  Be sure to include a slash at the end of the path to indicate that *logs* is a directory and not a file. When you make a change to the server's CLASSPATH, you must restart the server.

- Some servers read the <security-permission-spec> element of the resource adapter's deployment descriptor (the ra.xml file) and dynamically grant the resource adapter the security permissions listed in those elements.

  If you are executing a resource adapter-based application and experience a java.xxx.xxxPermission Exception, you have to manually add the contents of the <security-permission-spec> elements to the server's policy file. Consult the server's documentation for the location and format for editing the policy file. You should be able to simply copy and paste the elements into the server's policy file. Any changes to the policy generally require a server restart to take effect.

  If you make the changes and still experience Permission Exceptions, you might need to move some of the permission elements that you copied from the *resource adapter* domain in the policy file to the *default domain* in the policy file. This is because the resource adapter classes, especially if present in the server's CLASSPATH, might reside in the *default domain* and not the *resource adapter* domain.

CHAPTER 13

Understanding jdeinterop.ini for Java Connector

This chapter discusses the jdeinterop.ini file for the Java and dynamic Java connectors.

Settings for the jdeinterop.ini File for the Java Connector

The jdeinterop.ini file includes settings the server might need. The default location for the file is `c:\`; however, you can configure this location. This section provides details about the jdeinterop.ini file settings for the Java and dynamic Java connectors. Information is organized by section, for example [JDENET].These settings are discussed:

- OCM
- Cache
- JDENET
- Server
- Security
- Interop
- Events

Note. Unless otherwise indicated, the sections and settings are for all JD Edwards EnterpriseOne releases.

When you use Java interoperability connectors, you must also set up jdbj.ini file sections.

See Also

JD Edwards EnterpriseOne Tools 8.96 HTML Web Server Installation, Appendix D: Parameter and Values for the jdbj.ini File

[OCM]

Configure this [OCM] setting for the dynamic Java connector:

Setting and Typical Value	Purpose	Applicable Release
OCMEnabled=True	Selects or clears OCM inside the dynamic Java connector. A value of *true* indicates turned on.	All

[CACHE]

Configure these [CACHE] settings for the dynamic Java connector:

Setting and Typical Value	Purpose	Applicable Release
UserSession=0	Time out value (in milliseconds) for the dynamic Java connector user session. A zero (0) indicates infinite time out.	All
SpecExpire=30000000	Maximum time (in milliseconds) that the dynamic Java connector keeps the fetched spec in the cache.	All

[JDENET]

Configure these [JDENET] settings for the Java and dynamic Java connectors:

Setting and Typical Value	Purpose	Applicable Release
enterpriseServerTimeout= 90000	Timeout value for a request to the JD Edwards EnterpriseOne enterprise server.	All
maxPoolSize=30	JDENET socket connection pool size.	All
serviceNameConnect=6004	Port number used by the JD Edwards EnterpriseOne security server.	All

[SERVER]

Configure these [SERVER] settings for Java and dynamic Java connectors:

Setting and Typical Value	Purpose	Applicable Release
glossaryTextServer=JDED: 6010	The JD Edwards EnterpriseOne enterprise server and port that provide glossary text information.	All
codePage=1252	The encoding scheme, such as: 1252 English and Western European. 932 Japanese. 950 Traditional Chinese. 936 Simplified Chinese. 949 Korean.	All

[SECURITY]

Configure these [SECURITY] settings for Java and dynamic Java connectors:

Setting and Typical Value	Purpose	Applicable Release
NumServers=1	Number of security servers set.	All
SecurityServer=JDED	The JD Edwards EnterpriseOne security server.	All

[INTEROP]

Configure these INTEROP] settings for Java and dynamic Java connectors:

Setting and Typical Value	Purpose	Applicable Release
SettingTime=60000	Enables the connector to access and retrieve event information from the F90703 and F90704 tables. Defines the time for the connector applications to start up before the connector starts recovering an event. This value is milliseconds.	JD Edwards EnterpriseOne Tools 8.95 and later Tools releases with JD Edwards EnterpriseOne Applications 8.9 and later Applications releases. JD Edwards EnterpriseOne Tools 8.94 and JD Edwards EnterpriseOne Applications 8.11.
RecoveryInterval=10000	Enables the connector to access and retrieve event information from the F90703 and F90704 tables. Defines the time for the connector applications to start up before the connector starts recovering an event. This value is milliseconds.	JD Edwards EnterpriseOne Tools 8.95 and later Tools releases with JD Edwards EnterpriseOne Applications 8.9 and later Applications releases. JD Edwards EnterpriseOne Tools 8.94 with JD Edwards EnterpriseOne Applications 8.11.
enterpriseServer=JDED	The JD Edwards EnterpriseOne server.	All
port=6010	The port number of the JD Edwards EnterpriseOne server.	All
manual_timout=300000	The time-out value for a transaction in manual commit mode.	All
Repository=c:\jdedwards\Interop\repository	Points to the location of the repository directory containing business object libraries (generated JAR files).	All

[EVENTS]

Configure these [EVENTS] settings for Java and dynamic Java connectors:

Setting and Typical Value	Purpose	Applicable Release
UseGuaranteedEvents System=True	Indicates guaranteed event delivery using JD Edwards EnterpriseOne Tools 8.95 or later Tools release with JD Edwards EnterpriseOne Applications 8.9 or later Applications release. Values are true and false.	JD Edwards EnterpriseOne Tools 8.95 and later Tools releases with JD Edwards EnterpriseOne Applications 8.9 and later Applications releases.
Transport=HTTP	Defines the event transport mechanism. Valued values are HTTP and JMS. The default value is HTTP.	JD Edwards EnterpriseOne Tools 8.95 and later Tools releases with JD Edwards EnterpriseOne Applications 8.9 and later Applications releases. JD Edwards EnterpriseOne Tools 8.94 with JD Edwards EnterpriseOne Applications 8.11.
eventServiceURL= eventServiceURL=http: //hpdev1:9081/e1events /EventClientService	Locates the event service. If the value for the Transport= setting is HTTP, then this setting is configured.	JD Edwards EnterpriseOne Tools 8.95 and later Tools releases with JD Edwards EnterpriseOne Applications 8.9 and later Applications releases. JD Edwards EnterpriseOne Tools 8.94 with JD Edwards EnterpriseOne Applications 8.11.
jndiProviderURL=corbaloc:: denmlps14.mlab.jdedwards. com:9810 /NameServiceServer Root	Locates the event service. If the value for the Transport= setting is JMS, then this setting is configured.	JD Edwards EnterpriseOne Tools 8.95 and later Tools releases with JD Edwards EnterpriseOne Applications 8.9 and later Applications releases. JD Edwards EnterpriseOne Tools 8.94 with JD Edwards EnterpriseOne Applications 8.11.
port=6002	The socket port number where the EventListener receives the events from the JD Edwards EnterpriseOne server. This port should not be used by any other resource. Also, the port should not be changed dynamically when the connector is running, as this causes subsequent subscriptions to be lost.	All
ListenerMaxConnection=10	The maximum number of connections allowed by the EventListener. The default number of connections is 10, but you can change this number. The maximum number of connections allowed is 64.	All

Setting and Typical Value	Purpose	Applicable Release
ListenerMaxQueueEntry=10	The maximum number of events that the EventListener can hold before processing by the EventManager. The default number of events for the queue is 10, but you can change this number. The maximum number of events that can be held in the queue is 100.	All
Outbound_timeout=1200000	Maximum number of milliseconds that the EventManager waits before unsubscribing the transient event from the JD Edwards EnterpriseOne server.	All

CHAPTER 14

Understanding jdelog.properties File

This chapter discusses the settings for the jdelog.properties file.

Settings for the jdelog.properties File

The logging utility in the dynamic Java connector, the Java connector, and Java connector Architecture (JCA) is built on top of Apache Open Source Project Log4j. The jdelog.properties file defines the settings for the logging configuration. The jdelog.properties file should be physically located in CLASSPATH.

The jdelog.properties File consists of three log files:

- [E1LOG]
- [LOG1]
- [LOG2]

The following table provides a description of the parameters in each of the log files:

Parameter	Description
FILE	Set this value to the location of the log file.
LEVEL	Set this value to one of the following: • SEVERE • WARN • APP • DEBUG **Note.** The levels are listed above in the order of their priority, with SEVERE being the highest priority and DEBUG being the lowest priority. The default setting is APP.
FORMAT	Set this value to one of the following: • APPS • TOOLS • TOOLS_THREAD **Note.** In a Production environment, the FORMAT parameter should be set to APPS.
MAXFILESIZE	Set this value to the maximum file size of the log file. The default setting is 10MB. System performance can be affected if this value is set too high.

Parameter	Description
MAXBACKUPINDEX	Set this value to the maximum number of backups that need to be maintained. The default value is 20. System performance can be affected if this value is set too high.
COMPONENTS	Identify the components that need to be logged in the file. Components that you might use with a Java connector for interoperability include: JDBC, RUNTIME, INTEROP, JDBJ, PSFT, EVENTPROCESSOR.

The Server and Workstation Administration guide provides information for creating and managing jdelog.properties files. The HTML Web Server Installation guide provides a sample jdelog.properties file.

See JD Edwards EnterpriseOne Tools Server and Workstation Administration guide, Working with Server Administration Workbench, Using Web Client Logging with sawJAS

See JD Edwards Tools 8.96 HTML Web Server Installation Guide, Configuring JAS Logging, Sample jdelog.properties File

[E1LOG]

This is the section name for the root log. The following sample configuration logs all SEVERE and WARN messages to the jderoot.log file on your C drive.

[E1LOG]

FILE=C:\\ConnectorEventsClient\\log\\jderoot.log
LEVEL=WARN
FORMAT=APPS
MAXFILESIZE=10MB
MAXBACKUPINDEX=20
COMPONENT=ALL
APPEND=TRUE

[LOG1]

Logging RUNTIME and INTEROP components at the APP level is helpful for application developers. Application developers can use this log to analyze the flow of events in the web client. The following sample configuration logs all SEVERE, WARN, and APP messages to the jas.log file on your C drive.

[LOG1]

FILE=C:\\ConnectorEventsClient\\log\\jas.log
LEVEL=APP
FORMAT=APPS
MAXFILESIZE=10MB
MAXBACKUPINDEX=20
COMPONENT=RUNTIME | INTEROP | JDBJ
APPEND=TRUE

[LOG2]

Logging RUNTIME and INTEROP components at the DEBUG level is helpful for tools developers. Tools developers can use this log to debug tool level issues.

[LOG2]


```
FILE=C:\\ConnectorEventsClient\\log\\jasdebug.log
LEVEL=DEBUG
FORMAT=TOOLS_THREAD
MAXFILESIZE=10MB
MBMAXBACKUPINDEX=20
COMPONENT=RUNTIME | INTEROP | JDBJ
APPEND=TRUE
```


CHAPTER 15

Understanding iJDEScript

This chapter discusses iJDEScript and provides the iJDEScript commands.

iJDEScript

GenCOM and GenJava use a scripting language called iJDEScript that enables you to script code generation activities. Other than a few small differences, the scripting language is the same for these generators. You can use iJDEScript to:

- Rename business function libraries or select different business functions to create a custom interface; for example:

  library MyTestLibrary

  interface MytestInterface

  import B4200310 F4211FSEditLine

  import B000042

  This example selects the single business functions B4200310 F4211FSEditLine and B000042 for exposure.
- Use JD Edwards EnterpriseOne object aliases for more meaningful names.
- Select business functions to expose; for example:

  library MyAnotherLibrary

  importlib CAEC

  importlib CRUNTIME 1

  This example selects all of the business functions in the CAEC and CRUNTIME 1 libraries for exposure.

iJDEScript scripts have a simple syntax:

```
# comments begin with # and proceed to the end of line
# whitespace is ignored
login
importlib CAEC
build
```

iJDEScript Commands

iJDEScript supports a standard set of commands. These commands vary slightly for GenCOM and GenJava. These variations are indicated in these command descriptions:

Build Command

The build command tells the generator to generate code for all defined interfaces and to build the appropriate libraries.

When the build command is complete, the interface definitions are released. Using the build command again only generates code for interfaces defined after the last build command.

Syntax

This is an example of the syntax:

```
build
```

Call Command

The call command tells the generator to evaluate a subroutine with the given parameters. Parameters appear within the subroutine in order as special macros named %1%, %2%, and so on.

Syntax

This is an example of the syntax:

```
call sub [param [...]]
```

Example

This is an example:

```
login
call GenerateLib CAEC
call GenerateLib CALLBSFN
build
logout
```

Define Command

The define command tells the generator to optionally define a macro expansion. The value is expanded first, and then stored as the expansion of macro name. If name already has an expansion, the generator ignores this command.

Syntax

This is an example of the syntax:

```
define name value
```

Example

This is an example:


```
define val1 This is a test
define val2 %val1%!
define val2 This is ignored
say %val2%
generates the output
This is a test
```

Define! Command

The define! command tells the generator to define a macro expansion. The value is expanded first, and then stored as the expansion of macro name. If name already has an expansion, the generator replaces the current expansion with the new expansion.

Syntax

This is an example of the syntax:

```
define name value
```

Example

This is an example:

```
define val1 This is a test
define val2 %val1%!
define! val2 This is not ignored
say %val2%
generates the output
This is not ignored
```

Exit Command

The exit command tells the generator to exit the current subroutine or command file.

Syntax

This is an example of the syntax:

```
exit
```

Help Command

The help command requests help information from the generator on all available commands. Syntax information and a brief description are presented for each command. If command is specified, only help for command is provided.

Syntax

This is an example of the syntax:

```
help [command]
```

Import Command

The import command tells the generator to retrieve the specification of a function or group of business functions from the database and add them to the current interface definition. If only the business function name is specified, all functions from the specified business-function are retrieved and added to the current interface definition. If a function name is specified, only that function is retrieved and added to the current interface definition.

The alias option enables you to rename the function within the interface definition. The implementation still uses the original name when invoking the business function; however, the function is exposed as name through the interface.

Syntax

This is an example of the syntax:

```
import business-function [function [alias name]]
```

Example

This is an example:

```
library General
interface ReleaseMgmt
# Load GetReleaseAndVersion from B9800890; call it GetRV in
# ReleaseMgmt
import B4200310 F4211FSEditLine alias GetRV
# Load all functions from B000042
import B000042
```

Importlib Command

The importlib command tells the generator to import all business functions from the specified JD Edwards EnterpriseOne library, such as CAEC or CALLBSFN, into the current library definition. Each business function group results in the definition of an interface with the same name as the business function group and exposes as methods the functions within that group.

The category parameters enables you to restrict the import to one or more specific categories (1, 2, 3 and -; see the /Cat command line option).

Syntax

This is an example of the syntax:

```
importlib library [category [...]]
```

Example

This is an example:

```
library JDECOMInterfaceCAECCat1
# Load all category 1 functions from CAEC
importlib CAEC 1
build
```


Interface Command

The interface command tells the generator to begin the definition of an interface. All business functions retrieved using subsequent import commands become members of this interface.

Syntax for COM

This is an example of the syntax:

```
interface interface [ProgID prog-id] [vi-prog-id]
```

COM Example

This is an example:

```
interface ReleaseMgmt ProgID SOA.ReleaseMgmt.5 SOA.ReleaseMgmt
import B4200310 F4211FSEditLine
```

Library Command

The library command tells the generator that subsequent interface and import commands will generate definitions that belong in the library (DLL) named *name*. If the parameterset tag is also supplied, the library is used solely for parameterset definitions.

Note. When the library command without the parameter set tag is evaluated, parametersets for subsequent interface and import commands appear in that library until a library command with the parameterset tag is evaluated.

Syntax

This is an example of the syntax:

```
library name [parameterset]
```

Example

This is an example:

```
library Lib1
library Lib1Params parameterset
# Parametersets for CALLBSFN go in Lib1Params, but the
# business function interfaces go in Lib1
importlib CALLBSFN 2 3
```

Login Command

The login command tells the generator to log on to JD Edwards EnterpriseOne. If user, password, environment, and role are not specified, the user is prompted for the information.

Syntax

This is an example of the syntax:

```
login [user password environment role]
```


Example

This is an example:

```
login me mypassword demo
```

Logout Command

The logout command tells the generator to log off of JD Edwards EnterpriseOne.

Syntax

This is an example of the syntax:

```
logout
```

Opt Command

The opt command tells the generator to set the value of a generator command line parameter. The option parameter should not begin with the usual /. The value parameter does not undergo macro expansion.

Syntax

This is an example of the syntax:

```
opt option value
```

Example

This is an example:

```
# Do not generate business function interfaces, only
# parameterset interfaces
opt NoBSFN
```

Rename Command

The rename command tells the generator to rename an interface or a method within an interface. If a method is renamed, the correct business function is still called to build the implementation, but the method is exposed through the interface with a different name.

Syntax

This is an example of the syntax:

```
rename interface new
rename interface method new
```

Example

This is an example:

```
library Lib1
importlib CALLBSFN
rename B000042 BatchControl
rename BatchControl FSOpenBatch Open
```

```
rename BatchControl FSCloseBatch Close
```

Say Command

The say command tells the generator to display a message on the console.

Syntax

This is an example of the syntax:

```
say message
```

Example

This is an example:

```
say This is a test (%OwRelease%)
generate the output
This is a test (B9)
```

Sub Command

The sub command creates a subroutine definition. The call command may be used to invoke the subroutine. Parameters passed to the subroutine are as special macros named %1%, %2%, and so on.

Syntax

This is an example of the syntax:

```
sub name
   commands
end
```

Example

This is an example:

```
sub GenerateLibrary
      define source %1%
      library JDECOMInterface%source%Cat1
   importlib %source% 1
# Create a library of category 2 business functions in source
      opt NoBSFN
      library JDECOMInterface%source%Cat2
      importlib %source% 2
# Create a library of category 3 business functions in source
      library JDECOMInterface%source%Cat3
      importlib %source% 3
      system del /q c:\temp\*.*
      build
# Move the libraries to a staging area
      system mkdir d:\build
      system mkdir d:\build\Cat1
   system mkdir d:\build\Cat2
```

```
   system mkdir d:\build\Cat3
   system move JDECOMInterface%source%Cat1.* d:\build\Cat1
   system move JDECOMInterface%source%Cat2.* d:\build\Cat2
   system move JDECOMInterface%source%Cat3.* d:\build\Cat3
end
call GenerateLibrary CAEC
```

System Command

The system command tells the generator to evaluate a command in the shell.

Syntax

This is an example of the syntax:

```
system command
```

Example

This is an example:

```
say This is a test
generates the output
This is a test
```

APPENDIX A

Using the COM Connector Solution for Classic Events

This chapter provides an overview of COM connector events and discusses how to:

- Set up the COM server for Classic Events.
- Register components.
- Subscribe to events.
- Log COM events.
- Implement JD Edwards EnterpriseOne interfaces.
- Register EventSink for persistent subscription.

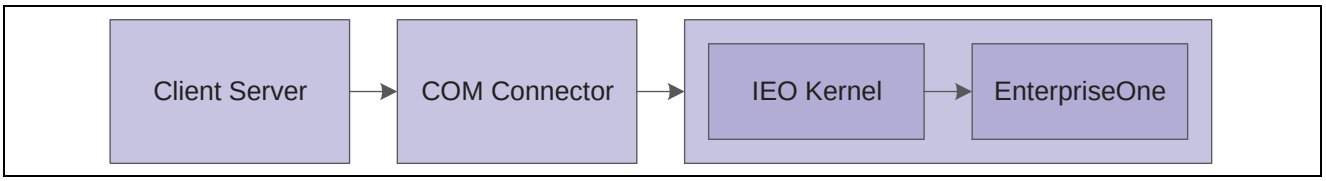
Note. This chapter is applicable only if you use classic event delivery. Classic event delivery is available if you use JD Edwards EnterpriseOne 8.93 or earlier releases, or if you use JD Edwards EnterpriseOne Tools 8.94 with JD Edwards EnterpriseOne Applications 8.10 or 8.9.

Refer to the Guaranteed Events chapters if you use JD Edwards EnterpriseOne Tools 8.94 with JD Edwards EnterpriseOne Applications 8.11 or JD Edwards EnterpriseOne Tools 8.95 or a later Tools release with JD Edwards EnterpriseOne Applications 8.9 or a later Applications release.

Understanding COM Connector Classic Events

The COM connector events solution uses Microsoft's COM+ Events Service. COM+ Events Loosely Coupled Events, which matches and connects publishers and subscribers, is part of the Microsoft Windows 2000 Component Services. The EventClass is a COM+ component that contains interfaces and methods that are used by the publisher to initiate events. The EventClass manages the connection between publisher and subscribers. The EventClass.dll, which contains the IOWEvent interface, is provided. The COM servers and COM clients must implement this interface so that when an event is initiated, this interface is called by the COM+ Events Service and the implementation is executed. The implementation decides what the delivered event and the event data should do. This implementation is COM server or COM client specific.

This illustration shows the COM connector architecture for classic events:



COM Connector Architecture for Classic Events (Software releases prior to JD Edwards EnterpriseOne Tools 8.94)

To support classic event delivery (JD Edwards EnterpriseOne Tools releases prior to 8.94), the COM connector uses the IEO kernel.

Note. You should have a basic understanding of the COM+ Events Service.

COM+ events supports Z events, real-time events, and XAPI events. COM+ Events Service is not dependent on JD Edwards EnterpriseOne setup for event generation.

See Also

Microsoft MSDN, http://msdn.microsoft.com/

JD Edwards EnterpriseOne Tools 8.96 Interoperability Guide, "Classic Events," Understanding Classic Events

JD Edwards EnterpriseOne Tools 8.96 Interoperability Guide, "Using Classic Real-Time Events," Understanding Real-Time Events - Classic

JD Edwards EnterpriseOne Tools 8.96 Interoperability Guide, "Using Classic XAPI Events," Understanding XAPI Events - Classic

Setting Up the COM Connector for Classic Events

This section provides an overview of the process for setting up the COM connector and discusses how to install and set up the COM connector to receive classic events.

Understanding COM Connector Set Up for Classic Events

These steps provide information for installing the COM connector so that you can receive classic events. Once the COM connector is installed, you set up the COM connector. Setting up the COM server includes setting up security and setting up the identity as an interactive user. After you install and set up the COM connector, you set up a DCOM server on a JD Edwards EnterpriseOne server machine. DCOM enables COM objects in a distributed environment. To ensure that the interoperability client works properly, you must set up DCOM for both a server environment and for a client environment.

Installing and Setting Up the COM Connector for Classic Events

Use these steps to install and set up the COM connector so that you can receive classic events.

Note. All of the COM connector required files will be installed with the JD Edwards EnterpriseOne client. If you have the JD Edwards EnterpriseOne client, ignore Step 1 and start with Step 2. If you do not have the JD Edwards EnterpriseOne client and you want to set up the COM connector on a third-party machine, start with Step 1.

1. Copy these files from the JD Edwards EnterpriseOne server (system\bin32) to a directory on the desired machine.

   For example, copy the files in c:\program files\JD Edwards to a non-JD Edwards EnterpriseOne client machine.

   - JDECOMConnector2.exe
   - JDECOMMN.dll
   - callobject.dll
   - comlog.dll

- EventManager.dll
- OneWorldInterfaceTx.dll
- xmlinterop.dll
- jdel.dll
- jdethread.dll
- jdeunicode.dll
- ustdio.dll
- icuil8n.dll
- jdeinterop.ini to c:\(root directory)
- checkver.exe
- ICUUC.dll
- Icu\data*.*
- IXXML4C2_3.dll
- EventClass.dll
- EventListener.dll

2. Create a new directory called Icu\data\ on the machine where the COM server is located.

   Copy all of the files from the JD Edwards EnterpriseOne server in folder system\Locale\xml*.* into Icu\data\. Create a new system variable, ICU_DATA, in the environment variables of the system properties and specify the path to the Icu\data\ as the value.

3. Use these steps to register the COM connector:
   a. Run this command

      ```
      c:\programfiles\JDEdwards\JDECOMConnector2.exe /RegServer
      ```

   b. Go to c:\programfiles\jdedwards\ Or c:\b9\system\bin32 and run these commands:

      ```
      regsvr32 EventManager.dll
      regsvr32 EventClass.dll
      ```

4. Create the JDEinterop.ini file by setting the JD Edwards EnterpriseOne server and port values to the JD Edwards EnterpriseOne application server with which you want the COM server to communicate.

   The COM server is now ready.

5. Use these steps to set up security on the COM server:
   a. From the Start menu, select Run.
   b. Enter Dcomcnfg.exe.
   c. On Distributed COM Configuration Properties, click the Default Security tab.
   d. Click the Edit Default Button in Default Access Permissions group.

      The Registry Value Permissions form appears. Some entries might already be present.
   e. On Registry Value Permissions, click Add.
   f. On Add Users and Groups, select the appropriate domain from the List Names From option.
   g. Click Everyone, and then click Add.

Type of access should be Allow Access.

h. Click OK.

No setup is required for default configuration permissions.

6. Use these steps to set up the identity as interactive user:

a. Run DCOMCnfg.

b. On Distributed COM Configuration Properties, select JDECOMConnector2, and then click Properties.

c. On JDECOMConnector2Properties, click the Identity tab, and then select the interactive user option.

d. Click Apply to apply the change.

Note. Every time you register the connector, you must set up the identity as an interactive user. If you copy the JDECOMConnector2.exe using Explorer, Explorer reruns the registration, and you must set up the identity as an interactive user.

To use Callbacks (Connection Points) with the COM solution, repeat these steps for setting up the identity as an interactive user on the COM client machine. Most of the shipped examples use Callbacks and require that you open the security on the client machine.

---

7. Use these steps to set up DCOM for a client environment:

a. From a DOS prompt on the DCOM client machine, run jdecomconnector2.exe /RegServer.

b. At the prompt, enter oleview.exe.

c. From the menu bar, select oleview.

d. Click View and select Expert Mode.

e. In the oleview window under Object Classes, double-click All Objects, and wait for all objects to appear.

f. Under All Objects, find and click Connector Class.

g. Click the Implementation tab on the right-side panel, and then click the local server and remove anything that appears in the editing window.

h. On the Activation tab, select the Launch as Interactive User option.

i. In Remote Machine Name, enter the COM server machine name.

j. Repeat steps 5 through 8 for MathNumeric Class.

The COM connector is installed and set up. You can start the DCOM client application.

Registering Components

So that subscribers can find an event class and subscribe to it, the JD Edwards EnterpriseOne event class must be registered with COM+. In addition, COM+ requires a type library that describes the event interface and methods so that subscribers and publishers can be properly matched and connected. The type library must reside in or be accompanied by a self-registering DLL.

To register the JD Edwards EnterpriseOne Events Class with COM+ Services, you must:

- Add a new COM+ application for the JD Edwards EnterpriseOne event class.
- Install the JD Edwards EnterpriseOne event class.

Note. Before you register the JD Edwards EnterpriseOne Event Class with COM+ Services, set up the COM server. The COM server can be set up on either a JD Edwards EnterpriseOne machine or a non-JD Edwards EnterpriseOne machine (third-party machine), or both.

See Also

Chapter 4, "Deploying the COM Solution for Business Function Execution," Installing COM Connector, page 24

Subscribing to Events

The COM connector supports both persistent and transient event subscriptions from the JD Edwards EnterpriseOne server. The events are subscribed from the JD Edwards EnterpriseOne server that is specified in the [INTEROP] section of the jdeinterop.ini file. The events are received through the EventListener. The EventListener runs as long as the COM connector is up and running. The COM connector runs as a small globe in the bottom right corner of the Microsoft Windows taskbar.

You must also set up the [EVENTS] section of the jdeinterop.ini file.

Note. The COM connector does not support subscription of events from multiple JD Edwards EnterpriseOne servers.

Logging COM Events

Logging for COM events is entered in the interopDebug.log file. The error log is interop.log.

Implementing JD Edwards EnterpriseOne Interfaces

This section discusses how to:

- Implement a JD Edwards EnterpriseOne interface.
- Create a COM+ component.
- Log on to the COM connector.
- Subscribe to an event.
- Add a new application.
- Install the event class.

Implementing a JD Edwards EnterpriseOne Interface

You must develop an object that implements the IOWEvent interface. For further discussion and for code samples in this document, the name EventSink is used as the object name. The object that you develop to implement the IOWEvent can have a different name. EventSink implements the IOWEvent interface and the method within the interface, and then consumes the JD Edwards EnterpriseOne event. The EventSink implementation is client specific. EventSink receives the event from JD Edwards EnterpriseOne by implementing the interface specified in EventClass.

This outline shows how to develop an EventSink component:

```
Option Explicit
Implements IOWEvent
Public Event OneWorldEvent(ByVal EventName As String, ByVal Data As String)

Public Sub IOWEvent_OneWorldEvent(ByVal EventName As String, ByVal Data
As String)
'// Add code specific to the client implementation here
    RaiseEvent OneWorldEvent(EventName, Data)
End Sub
```

This list outlines the steps for you to follow to use the EventManager library and MessageHandler Interface to subscribe to events.

1. Log on to the connector.

   Successful logon returns an access number.
2. Create the EventSink object.
3. Create the MessageHandler object.
4. Call methods on the MessageHandle for Subscribe, Unsubscribe, GetTemplate, and GetEventList for the respective event.
5. To keep the session alive and not time out from receiving events, call the UpdateOutBoundSessionTime method on the connector interface.

   This method updates the user session time to the current time.
6. To subscribe to the events as persistent, register VB EventSink in the COM+ Component Services and add the subscription for the EventClass.

Creating a COM+ Component

This sample code is for creating a COM+ component named EventSink.dll. EventSink implements the EventClass interface IOWEvent(). You can use a name other than EventSink.

EventSink: OneWorldTransientEventSink.cls

This is the sample code for creating a COM+ component:

```
Option Explicit

Implements IOWEvent
Public Event OneWorldEvent(ByVal EventName As String, ByVal Data As
```



```
String)

Public Sub IOWEvent_OneWorldEvent(ByVal EventName As String, ByVal
Data As String)
   Dim flsObject As New Scripting.FileSystemObject
   Dim varEventFile As TextStream
   Dim strEventFile As String
   strEventFile = "C:\temp\eventDataPer.xml". ' change this to a
valid directory
   If Dir(strEventFile) = "" Then Set varEventFile =
      flsObject.CreateTextFile(strEventFile, False, False)
   Else
       Set varEventFile = flsObject.OpenTextFile(strEventFile,
ForWriting, False)
   End If

   varEventFile.WriteLine Data
   varEventFile.Close
   RaiseEvent OneWorldEvent(EventName, Data)
End Sub
```

Logging on to the COM Connector

This sample code logs on to the COM connector, creates the MessageHandler object, and performs Subscribe, Unsubscribe, GetTemplate, and GetList. Before executing the subscriber, use the Regsvr32 command to register COMConnector.dll.

COMConnector: frmLogin.frm

This code sample shows logging on to the COM connector:

```
Option Explicit

Public bLoginEnv As Boolean

Private Sub cmdCancel_Click()
    'set the global var to false
    'to denote a failed login
    bLoginEnv = False
    Me.Hide
End Sub

Private Sub cmdOK_Click()
    'check for correct password
    If txtUserName = "" Or txtenvironment = "" Then
        bLoginEnv = False
        MsgBox "Must Enter User Name and Environment to continue"
    Else
        bLoginEnv = True
        Me.Hide
    End If
```

```
End Sub
```

COMConnector Common.bas

This code sample shows creating the message handler:

```
Option Explicit
Dim conn As New Connector
Dim connRole As IConnector2
Dim messageHandler As New messageHandler
Dim mHandlerInterface As ImessageHandler
Dim lngAccessNumber As Long
Public Sub comm_Initialize()
    Set connRole = conn
    frmLogin.bLoginEnv = False
    frmLogin.Show
    While Not frmLogin.bLoginEnv
        DoEvents
    Wend
    lngAccessNumber = connRole.Login(frmLogin.txtUserName,
frmLogin.txtPassword, frmLogin.txtenvironment, frmLogin.txtrole)
    Set mHandlerInterface = messageHandler
End Sub

' NOTE: the code in this module is particular to this prototype.
' Different code would be used in a production version to send
' messages to JD Edwards EnterpriseOne using appropriate communication
' prototocols

Public Sub SendSubscriptionToOneWorld(eventName As String,
oneworldevent As IOWEvent, mode As Long)
    mHandlerInterface.SubscribeEvent lngAccessNumber, conn, eventName,
oneworldevent, mode
End Sub
Public Sub SendUnSubscribeToOneWorld(eventName As String,
oneworldevent As IOWEvent, mode As Long)
    mHandlerInterface.UnSubscribeEvent lngAccessNumber, conn,
eventName, oneworldevent, mode
End Sub
Public Sub getEventListFromOneWorld(eventList As String)
    mHandlerInterface.GetEventList lngAccessNumber, conn, eventList
End Sub
Public Sub getEventTemplateFromOneWorld(eventName As String,
eventTemplate As String)
    mHandlerInterface.GetEventTemplate lngAccessNumber, eventName,
conn, eventTemplate
End Sub
```

COMConnector: SubscriptionManager

This code sample shows event subscription and unsubscribe:


```
Option Explicit

Private m_SubscribedEvents As Collection

Private Sub Class_Initialize()
    Set m_SubscribedEvents = New Collection
    comm_Initialize
End Sub
Public Sub GetEventList(eventList As String)
    getEventListFromOneWorld eventList
End Sub

Public Sub CreateTransientSubscription(eventName As String,
oneworldevent As IOWEvent)
    SubscribeToOneWorldEvent eventName, oneworldevent, 0
End Sub
Public Sub CreatePersistentSubscription(eventName As String,
oneworldevent As IOWEvent)
    SubscribeToOneWorldEvent eventName, oneworldevent, 1
End Sub
Public Sub RemoveTransientSubscription(eventName As String,
oneworldevent As IOWEvent)
    UnSubscribeToOneWorldEvent eventName, oneworldevent, 0
End Sub
Public Sub RemovePersistentSubscription(eventName As String,
oneworldevent As IOWEvent)
    UnSubscribeToOneWorldEvent eventName, oneworldevent, 1
End Sub
Public Sub GetEventTemplate(eventName As String, eventTemplate As
String)
    getEventTemplateFromOneWorld eventName, eventTemplate
End Sub
Public Sub SubscribeToOneWorldEvent(eventName As String, oneworldevent
As IOWEvent, mode As Long)
'Private Function SubscribeToOneWorldEvent(EventName As String) As
'Boolean we've already subscribed if the subscription is in our list
    Dim alreadySubscribed As Boolean
    alreadySubscribed = (CollectionContainsString
    (m_SubscribedEvents,eventName) = True)

    ' now do the right thing...
    If (alreadySubscribed = False) Then
        ' this instance of the COMConnector has not seen this event
        ' before, so add it to our list...
        m_SubscribedEvents.Add (eventName)

        ' and go ahead and subscribe to the event from JD Edwards
        ' EnterpriseOne
        SendSubscriptionToOneWorld eventName, oneworldevent, mode
    End If
```

```
 'SubscribeToOneWorldEvent = alreadySubscribed
End Sub

Private Function CollectionContainsString(col As Collection, str As
String)
    Dim colItem As Variant
    For Each colItem In col
        If (colItem = str) Then
            CollectionContainsString = True
            Exit Function
        End If
    Next
    CollectionContainsString = False
End Function

Public Sub UnSubscribeToOneWorldEvent(eventName As String,
oneworldevent As IOWEvent, mode As Long)
Dim alreadySubscribed As Boolean
    alreadySubscribed = (RemoveFromCollection(m_SubscribedEvents,
eventName))
        If (alreadySubscribed = False) Then
            MsgBox "Event Not Subscribed"
        Else
            ' and go ahead and subscribe to the event from
            ' JD Edwards EnterpriseOne
            SendUnSubscribeToOneWorld eventName, oneworldevent, mode
        End If
  ' End If
End Sub
Private Function RemoveFromCollection(col As Collection, str As
String) Dim colItem As Variant
    Dim count As Integer
    count = 0
    For Each colItem In col
        count = count + 1
        If (colItem = str) Then
            col.Remove count
            RemoveFromCollection = True
            Exit Function
        End If
    Next
    RemoveFromCollection = False
End Function
```


Subscribing to Events

Subscriber is the GUI that gets the EventsList, EventTemplate, Subscribe, and Unsubscribe. Subscriber is built as a VB executable. Typical usage is to get the EventList first, which populates the option list with the events that are supported by the JD Edwards EnterpriseOne server. Select the event that needs to be subscribed from the JD Edwards EnterpriseOne server and the type of subscription. Click Subscribe to add a Subscription, or click Unsubscribe to unsubscribe from the JD Edwards EnterpriseOne server. The Subscribed events and the Received events are depicted in separate boxes. The received event is displayed in the window on the right. Before building the subscriber, you should use the Regsvr32 command to register EventSink.dll and COMConnector.dll.

Subscriber: MainForm.frm

This code sample is for the GUI and the control buttons on the GUI. This code should be built after registering the COMConnector.dll and MyEventSink.dll.

```
Option Explicit
' ------------------------------ ** ---------------------------------
'                            Member Variables
' ------------------------------ ** ---------------------------------
Private m_SubscriptionManager As SubscriptionManager
Private WithEvents m_OneWorldTransientEventSink As
OneWorldTransientEventSink
Private Sub Combo1_Change()

End Sub
Private Sub Check1_Click()

End Sub

Private Sub btnClear_Click(Index As Integer)
    lvwReceivedEvents.ListItems.Clear
End Sub
'----------------------------- ** -----------------------------------
'                         GetEventTemplate
'----------------------------- ** -----------------------------------
Private Sub btnGetEventTemplate_Click()
    Dim EventName As String
    Dim EventTemplate As String
    EventName = cEventList.List(cEventList.ListIndex)
    m_SubscriptionManager.GetEventTemplate EventName, EventTemplate
    Dim flsObject As New Scripting.FileSystemObject
    Dim varTemplateFile As TextStream
    Dim strTemplateFile As String
    strTemplateFile = "C:\temp\event_template.xml"
    If Dir(strTemplateFile) = "" Then
      Set varTemplateFile = flsObject.CreateTextFile(strTemplateFile
False, False)
    Else
      Set varTemplateFile = flsObject.OpenTextFile(strTemplateFile,
ForWriting, False)
    End If
```



```
    varTemplateFile.WriteLine EventTemplate
    varTemplateFile.Close

    wbEventData.Navigate "c:\temp\event_template.xml"
End Sub

' ------------------------------ ** ---------------------------------
'                             Event Handlers
' ------------------------------ ** ---------------------------------

Private Sub m_OneWorldTransientEventSink_OneWorldEvent(ByVal EventName
As String, ByVal Data As String)
    ' add the event name and payload to the list
    Dim mTempItem As ListItem
    Set mTempItem = lvwReceivedEvents.ListItems.Add()
    mTempItem.Text = EventName
    'mTempItem.SubItems(1) = Data
    Dim flsObject As New Scripting.FileSystemObject
    Dim varEventFile As TextStream
    Dim strEventFile As String
    strEventFile = "C:\temp\eventData.xml"
    If Dir(strEventFile) = "" Then
        Set varEventFile = flsObject.CreateTextFile(strEventFile,
False,False)
    Else
        Set varEventFile = flsObject.OpenTextFile(strEventFile,
ForWriting, False)
    End If

    varEventFile.WriteLine Data
    varEventFile.Close
    wbEventData.Navigate "c:\temp\eventdata.xml"
End Sub
'---------------------------- ** ----------------------------------
'                         GetEventList
'---------------------------- ** -   ---------------------------------
Private Sub btnGetEventList_Click()
    Dim events As String
    Dim myValue As String
    Dim myString As String
    Set m_SubscriptionManager = New SubscriptionManager
    m_SubscriptionManager.GetEventList events

    cEventList.Clear
    myString = events
    Do Until events = ""
        If InStr(1, myString, ":") > 0 Then
            myValue = Left(myString, InStr(1, myString, ":") - 1)
            myString = Mid(myString, InStr(1, myString, ":") + 1)
```



```
        Else
            myValue = myString
            events = ""
        End If

        cEventList.AddItem myValue
    Loop
    cEventList.ListIndex = 0
End Sub
'---------------------------- ** ----------------------------------
'                          Subscribe Event
'---------------------------- ** ----------------------------------
Private Sub btnSubscribe_Click()
    ' subscribe to the named event.
    Dim EventName As String
    EventName = cEventList.List(cEventList.ListIndex)
    If (chkPersist.Value = Checked) Then
      m_SubscriptionManager.CreatePersistentSubscription EventName,
m_OneWorldTransientEventSink
    Else
      m_SubscriptionManager.CreateTransientSubscription EventName,
m_OneWorldTransientEventSink
    End If
    Dim mTempItem As ListItem
    Set mTempItem = lvwSubscribedEvents.ListItems.Add()
    mTempItem.Text = EventName
End Sub
'---------------------------- ** ----------------------------------
'                         UnSubscribe Event
'---------------------------- ** ----------------------------------
Private Sub btnUnsubscribe_Click()
    Dim EventName As String
    EventName = cEventList.List(cEventList.ListIndex)
    Dim lstItem As ListItem
    Dim count As Integer
    Dim found As Boolean
    count = 0
    found = False
    For Each lstItem In lvwSubscribedEvents.ListItems
        count = count + 1
        If lstItem = EventName Then
            lvwSubscribedEvents.ListItems.remove (count)
            GoTo remove
            found = True
        End If
    Next
    If found = False Then
        MsgBox "Event Not Subscribed"
    End If
remove: If (chkPersist.Value = Checked) Then
```

```
      m_SubscriptionManager.RemovePersistentSubscription EventName,
m_OneWorldTransientEventSink
    Else
      m_SubscriptionManager.RemoveTransientSubscription EventName,
m_OneWorldTransientEventSink
    End If

End Sub
'---------------------------- ** ----------------------------------
'                        Clear the Received Events List
'---------------------------- ** ----------------------------------
Private Sub btnClear0_Click()
    ' clear the events from the list
    lvwReceivedEvents.ListItems.Clear
End Sub

Private Sub btnClose_Click()
    Unload Me
    End
End Sub
```

Adding a New Application

From a Microsoft Windows 2000 machine, navigate to COM+ Applications (Control Panel > Administrative Tools > Component Services), and then expand these buttons and folders:

Component Services > Computers > My Computer > COM+ Applications

To add a new application:

1. On Component Services, select COM+ Applications.
2. Right-click COM+ Applications, select New, and then select Application.

   The COM Application Install Wizard appears. These steps apply to the wizard.
3. On Install or Create a New Application, select Create an empty application.
4. On Create Empty Application, enter the name of the application (for example, JDECOMConnectorEvents).
5. Select an option for Activation Type, and then click Next.
6. On Set Application Identity, select the Interactive User option, and then click Next.
7. Click Finish.

A new application, with the name you entered in Step 4, is added to COM+ Applications.

Installing the Event Class

On Component Services, expand the folder for the new application (for example, JDECOMConnectorEvents).

To install the event class:

1. On Component Services, select Components.
2. Right-click Components, select New, and then select Component.

The COM Component Install Wizard appears. These steps apply to the wizard.

3. On Import or Install a Component, select Install new event class(es).
4. On Select Files to Install, browse to the EventClass.dll on the Windows 2000 machine.
5. Select EventClass.dll, and then click Open.

   Install new event class appears with information in these fields:

   - Files to install
   - Event classes found
6. Click Next, and then click Finish.

EventClass.dll is successfully added to Component Services.

Registering EventSink for Persistent Subscription

After you register an event class in the COM+ catalog, you can add subscribers to the event class and subscriptions to the subscribers. For persistent event subscription:

- Add a new application for EventSink.
- Install the type library component for EventSink.
- Add a subscription.

Note. To add EventSink, follow the steps in the task To add a new application in the Connectors Guide. The name of the application is EventSink, or a name that you prefer.

To install the EventSink component:

On Component Services, expand the folder for the new application (for example, EventSink).

1. Select Components.
2. Right-click Components, select New, and then select Component.

   The COM Component Install Wizard appears. These steps are for the wizard.
3. On Import or Install a Component, select Install new component(s).
4. On Select Files to Install, browse to the EventSink.dll that you previously developed.
5. Select EventSink.dll, and then click Open.

   Install new component appears with information in these fields:

   - Files to install
   - Event classes found
6. Click Next, and then click Finish.

EventSink.dll is successfully added to Component Services.

To add a subscription:

In COM+ Applications, expand these folders:

JDECOMConnectorEvents > Components > EventSink.OneWorldTransientEventSink

1. Select Subscription.
2. Right-click Subscription, select New, and then select Subscription.

   The COM New Subscription Wizard appears. These steps apply to the wizard.
3. On Select Subscription Method(s), chose IOWEvent, and then click Next.
4. If appropriate, select the Use all interfaces for this component option.
5. On Select Event Class, select the event class (for example, JDEdwards.EventClass.OneWorldEventClass.1), and then press Next.

   If multiple EventSink classes have implemented the event interface, then use all event classes that implement that specified interface. If only one EventSink class has implemented the event interface, then just select that specific class.
6. On Subscription Options, enter the name of the subscription (for example, MySubscription).
7. In the Options area, select the Enable this subscription immediately option, and then click Next.
8. Click Finish.

   A new subscription, with the name you entered in Step 6, is added to COM+ Services. You must define the name of the event for the subscription.
9. Right-click the subscription (for example, MySubscription), and then select Properties.
10. On MySubscription Properties, click the Options tab.
11. Chose the Enabled option.
12. In the Filter criteria field, enter the name of the event for which you want a subscription.

    Enter all of the events for which you want to subscribe. The filter criteria string supports relational operations (=, ==, !, !=, ~, ~=, <>), nested parentheses, and logical words (AND, OR, and NOT); for example:

    EventName==’RTSOOUT’ OR EventName==RTPOOUT’
13. Click OK.

APPENDIX B

Using the Java Connector Solution for Classic Events

This chapter provides an overview of Java connector events and discusses how to develop the Java client to use the Java connector event source.

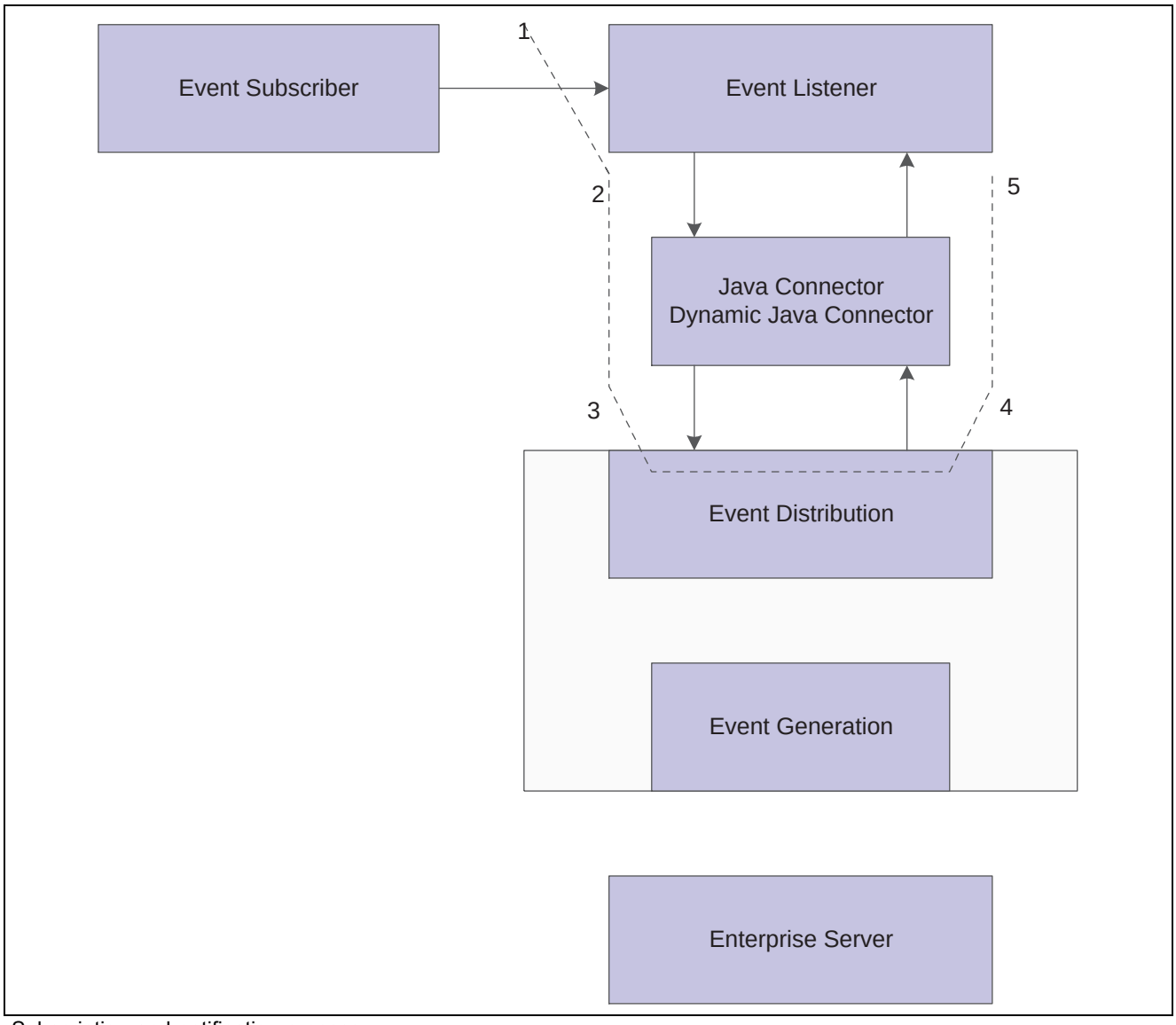
Note. This chapter is applicable only if you use classic event delivery. Classic event delivery is available when you use JD Edwards EnterpriseOne 8.93 or earlier releases, or if you use JD Edwards EnterpriseOne Tools 8.94 with JD Edwards EnterpriseOne Applications 8.10 or earlier releases of the JD Edwards EnterpriseOne Applications.

Refer to the Guaranteed Events chapters if you use JD Edwards EnterpriseOne Tools 8.94 with JD Edwards EnterpriseOne Applications 8.11 or JD Edwards EnterpriseOne Tools 8.95 or a later Tools release with JD Edwards EnterpriseOne Applications 8.9 or a later Applications release.

Understanding Java Connector Events

The Java connector outbound event source architecture enables Java clients to use either the Java connector or the dynamic Java connector to subscribe to various transaction types in JD Edwards EnterpriseOne and receive notification upon completion of those transactions. For example, a client can subscribe to the event JDESOOUT and then receive notification when a sales order transaction is complete in JD Edwards EnterpriseOne.

This diagram illustrates the subscription and notification process:



Subscription and notification process

1. JD Edwards EnterpriseOne clients create different types of EventListeners.
2. JD Edwards EnterpriseOne clients subscribe to various event types with the Java connector.
3. When the Java connector receives a subscription for a given event, it subscribes to the same event type with the event distribution kernel.
4. JD Edwards EnterpriseOne events originate from the real-time events kernel or from callback functions in Uses.

   When the event distribution kernel receives an event to which the Java connector has subscribed, it sends the event to the Java connector.
5. The Java connector sends the event to all subscribers for that event.

   The EventListener callback function is executed to receive the subscribed event.

Note. The outbound events architecture is the same for the Java connector and the dynamic Java connector. The difference is that corresponding package location for the dynamic Java connector is comjdedwards.system.connector.dynamic.* and com.jdedwards.system.connector.dynamic.events.*.

For purposes of discussion in this document, the Java connector is used to illustrate the outbound events architecture. Special notes are added to discuss any API differences between the Java connector and dynamic Java connector.

Do not mix the usage of APIs from the two connectors in one application.

Developing the Java Client to Use the Java Connector Event Source

You use the Java connector outbound event source to subscribe to an outbound event. This list identifies the tasks for setting up and using the Java client to subscribe to JD Edwards EnterpriseOne transaction types and notify you upon completion of the transaction:

- Create a Java class to implement an interface.
- Create a Java client application to subscribe to an event.
- Compile the Java client.
- Run the Java client.

Creating a Java Class to Implement an Interface

You create a Java class to implement an interface to JD Edwards EnterpriseOne. Depending on the purpose for which you are using the Java class, implement one of these interfaces:

- com.jdedwards.system.connector.events.CountedListener

  Implement this interface if you want to know the subscription count, when the subscription count is reached, and when the subscribed event is dropped.

- com.jdedwards.system.connector.events.PersistentListener

  Implement this interface if you want the real-time event kernel to persist the subscription when the kernel goes down and comes up, and the connection to the Java connector is reestablished.

- com.jdedwards.system.connector.events.EventListener

  Implement this interface for most other situations.

No matter which interface you implement, the implementation Java class must contain these five methods:

```
//set the event type to subscribe
void setEventType( String type );
String getEventType();

//stop/start the event coming in the Java connector
void setPause( boolean pause );
boolean isPaused();

//the callback function when the event arrives
void onOneWorldEvent( EventObject event );
```

Creating a Java Client Application to Subscribe to an Event

You create a Java client application to subscribe to an event. The Java client application must:

1. Create a new instance of the Connector class.
2. Use the connector object to verify the client's user ID, password, and environment, and then log the client into JD Edwards EnterpriseOne.
3. Do one of these:
   - For Java connector, create an EventSource object by calling the CreateBusinessObject method of the Connector class, passing in an Events::EventSource string identifier.
   - For dynamic Java connector, Get EventSource instance by using this command:

     ```
     com.jdedwards.system.connector.dynamic.connector.events.EventSource.
     getInstance()
     ```
4. Create an EventListener object.
5. Specify the specific event type to which to subscribe.
6. Register the EventListener object with the EventSrc object.
7. Develop a callback function.

   When the subscribed to event arrives, the EventListener calls this callback function. This step is optional.

Example: Using the Java Client to Subscribe to an Event Using the Java Connector Outbound Event Source

This example illustrates how to write code for a Java client to subscribe to an event using the Java connector outbound event source.

```
import java.io.*;
import javax.swing.*;
import com.jdedwards.system.connector.*;
import com.jdedwards.system.connector.events.*;



... //Declare Class
{
 Connector m_connector = null;
 EventSource m_theSource = null;
 ListenerImpl_Page132 m_listener = null;
  try
  {
    m_connector = new Connector();
    int m_Access = 0;
  m_Access = m_connector.Login("user", "pwd", "env");

  // passing in an "Events::EventSource" string identifier.
  m_theSource = (EventSource)m_connector.CreateBusinessObject
```



```
("Events::EventSource", m_Access);

  m_listener = new ListenerImpl(this);

  m_listener.setEventType("JDESOOUT");

  m_theSource.addListener( m_listener, m_Access );
  }
  catch(Exception e)
  {
  System.out.println(e.toString());
  System.out.println(e.getMessage());
  e.printStackTrace();
  }
 }

 public synchronized void executeCallBack(EventObject event)
 {
  System.out.println("Getting the event:"+event.getData());
//execute the call back function;
 }
}

class ListenerImpl_Page132 implements EventListener
{
 String m_eventType;
 boolean m_paused = false;
 EventClient_Page132 m_client;
 /** Creates new Listener*/
 public ListenerImpl()
 { }
 public ListenerImpl(EventClient
 { this.m_client=client;
 }
 public synchronized String getEventType()
 {
 return m_eventType;
 }
 public void setEventType(java.lang.String eventType)
 {
 this.m_eventType = eventType;
 }
 public synchronized boolean isPaused()
 {
 return m_paused;
 }
 public synchronized void setPause(boolean pause)
 {
 m_paused = pause;
 }
```

```
public synchronized void onOneWorldEvent(EventObject p1)
{
System.out.println("Received event: " + p1.getType());
// if the arrival event is the one that client subscribes,
// the EventListner can trigger the call back function in the client
if (p1.getType().equalsIgnoreCase(m_eventType)) {
m_client.executeCallBack(p1);
}
}
```

Compiling the Java Client

To compile the Java client, use this command:

```
set JAVA_HOME = <the path of JDK>
set OneWorld_HOME = <the installation path>
set CLASSPATH=%OneWorld_HOME%\system\classes\base_JAR.jar
set CLASSPATH=%OneWorld_HOME%\system\classes\jdeNet_JAR.jar
set CLASSPATH=%OneWorld_HOME%\system\classes\system_JAR.jar
set CLASSPATH=%CLASSPATH%;%OneWorld_HOME%\system\classes\Connector.jar
set CLASSPATH=%CLASSPATH%;%OneWorld_Home%\system\classes\log4j.jar
set CLASSPATH=%CLASSPATH%;%OneWorld_HOME%\system\classes\xalan.jar
set CLASSPATH=%CLASSPATH%;%OneWorld_HOME%\system\classes\xerces.jar
%JAVA_HOME%\bin\javac -classpath %CLASSPATH% EventClient.java
EventListenerImpl
```

Running the Java Client

To run the Java client, use this command:

```
set JAVA_HOME = <the path of JDK>
set OneWorld_HOME = <the installation path>
set CLASSPATH=%OneWorld_HOME%\system\classes\base_JAR.jar
set CLASSPATH=%OneWorld_HOME%\system\classes\jdeNet_JAR.jar
set CLASSPATH=%OneWorld_HOME%\system\classes\system_JAR.jar
set CLASSPATH=%CLASSPATH%;%OneWorld_HOME%\system\classes\Connector.jar
set CLASSPATH=%CLASSPATH%;%OneWorld_Home%\system\classes\log4j.jar
set CLASSPATH=%CLASSPATH%;%OneWorld_HOME%\system\classes\xalan.jar
set CLASSPATH=%CLASSPATH%;%OneWorld_HOME%\system\classes\xerces.jar
%JAVA_HOME%\bin\java -classpath %cp%
EventClient
```


Glossary of JD Edwards EnterpriseOne Terms

activity	A scheduling entity in JD Edwards EnterpriseOne tools that represents a designated amount of time on a calendar.
activity rule	The criteria by which an object progresses from one given point to the next in a flow.
add mode	A condition of a form that enables users to input data.
Advanced Planning Agent (APAg)	A JD Edwards EnterpriseOne tool that can be used to extract, transform, and load enterprise data. APAg supports access to data sources in the form of rational databases, flat file format, and other data or message encoding, such as XML.
application server	A server in a local area network that contains applications shared by network clients.
as if processing	A process that enables you to view currency amounts as if they were entered in a currency different from the domestic and foreign currency of the transaction.
alternate currency	A currency that is different from the domestic currency (when dealing with a domestic-only transaction) or the domestic and foreign currency of a transaction. In JD Edwards EnterpriseOne Financial Management, alternate currency processing enables you to enter receipts and payments in a currency other than the one in which they were issued.
as of processing	A process that is run as of a specific point in time to summarize transactions up to that date. For example, you can run various JD Edwards EnterpriseOne reports as of a specific date to determine balances and amounts of accounts, units, and so on as of that date.
back-to-back process	A process in JD Edwards EnterpriseOne Supply Management that contains the same keys that are used in another process.
batch processing	A process of transferring records from a third-party system to JD Edwards EnterpriseOne. In JD Edwards EnterpriseOne Financial Management, batch processing enables you to transfer invoices and vouchers that are entered in a system other than JD Edwards EnterpriseOne to JD Edwards EnterpriseOne Accounts Receivable and JD Edwards EnterpriseOne Accounts Payable, respectively. In addition, you can transfer address book information, including customer and supplier records, to JD Edwards EnterpriseOne.
batch server	A server that is designated for running batch processing requests. A batch server typically does not contain a database nor does it run interactive applications.
batch-of-one immediate	A transaction method that enables a client application to perform work on a client workstation, then submit the work all at once to a server application for further processing. As a batch process is running on the server, the client application can continue performing other tasks. See also direct connect and store-and-forward.
business function	A named set of user-created, reusable business rules and logs that can be called through event rules. Business functions can run a transaction or a subset of a transaction (check inventory, issue work orders, and so on). Business functions also contain the application programming interfaces (APIs) that enable them to be called from a form, a database trigger, or a non-JD Edwards EnterpriseOne application. Business functions can be combined with other business functions, forms, event rules,

	and other components to make up an application. Business functions can be created through event rules or third-generation languages, such as C. Examples of business functions include Credit Check and Item Availability.
business function event rule	See named event rule (NER).
business view	A means for selecting specific columns from one or more JD Edwards EnterpriseOne application tables whose data is used in an application or report. A business view does not select specific rows, nor does it contain any actual data. It is strictly a view through which you can manipulate data.
central objects merge	A process that blends a customer's modifications to the objects in a current release with objects in a new release.
central server	A server that has been designated to contain the originally installed version of the software (central objects) for deployment to client computers. In a typical JD Edwards EnterpriseOne installation, the software is loaded on to one machine—the central server. Then, copies of the software are pushed out or downloaded to various workstations attached to it. That way, if the software is altered or corrupted through its use on workstations, an original set of objects (central objects) is always available on the central server.
charts	Tables of information in JD Edwards EnterpriseOne that appear on forms in the software.
connector	Component-based interoperability model that enables third-party applications and JD Edwards EnterpriseOne to share logic and data. The JD Edwards EnterpriseOne connector architecture includes Java and COM connectors.
contra/clearing account	A general ledger account in JD Edwards EnterpriseOne Financial Management that is used by the system to offset (balance) journal entries. For example, you can use a contra/clearing account to balance the entries created by allocations in JD Edwards EnterpriseOne Financial Management.
Control Table Workbench	An application that, during the Installation Workbench processing, runs the batch applications for the planned merges that update the data dictionary, user-defined codes, menus, and user override tables.
control tables merge	A process that blends a customer's modifications to the control tables with the data that accompanies a new release.
cost assignment	The process in JD Edwards EnterpriseOne Advanced Cost Accounting of tracing or allocating resources to activities or cost objects.
cost component	In JD Edwards EnterpriseOne Manufacturing, an element of an item's cost (for example, material, labor, or overhead).
cross segment edit	A logic statement that establishes the relationship between configured item segments. Cross segment edits are used to prevent ordering of configurations that cannot be produced.
currency restatement	The process of converting amounts from one currency into another currency, generally for reporting purposes. You can use the currency restatement process, for example, when many currencies must be restated into a single currency for consolidated reporting.
database server	A server in a local area network that maintains a database and performs searches for client computers.
Data Source Workbench	An application that, during the Installation Workbench process, copies all data sources that are defined in the installation plan from the Data Source Master and Table and Data Source Sizing tables in the Planner data source to the system-release number data source. It also updates the Data Source Plan detail record to reflect completion.

date pattern A calendar that represents the beginning date for the fiscal year and the ending date for each period in that year in standard and 52-period accounting.

denominated-in currency The company currency in which financial reports are based.

deployment server A server that is used to install, maintain, and distribute software to one or more enterprise servers and client workstations.

detail information Information that relates to individual lines in JD Edwards EnterpriseOne transactions (for example, voucher pay items and sales order detail lines).

direct connect A transaction method in which a client application communicates interactively and directly with a server application.

See also batch-of-one immediate and store-and-forward.

Do Not Translate (DNT) A type of data source that must exist on the iSeries because of BLOB restrictions.

dual pricing The process of providing prices for goods and services in two currencies.

edit code A code that indicates how a specific value for a report or a form should appear or be formatted. The default edit codes that pertain to reporting require particular attention because they account for a substantial amount of information.

edit mode A condition of a form that enables users to change data.

edit rule A method used for formatting and validating user entries against a predefined rule or set of rules.

Electronic Data Interchange (EDI) An interoperability model that enables paperless computer-to-computer exchange of business transactions between JD Edwards EnterpriseOne and third-party systems. Companies that use EDI must have translator software to convert data from the EDI standard format to the formats of their computer systems.

embedded event rule An event rule that is specific to a particular table or application. Examples include form-to-form calls, hiding a field based on a processing option value, and calling a business function. Contrast with the business function event rule.

Employee Work Center A central location for sending and receiving all JD Edwards EnterpriseOne messages (system and user generated), regardless of the originating application or user. Each user has a mailbox that contains workflow and other messages, including Active Messages.

enterprise server A server that contains the database and the logic for JD Edwards EnterpriseOne.

EnterpriseOne object A reusable piece of code that is used to build applications. Object types include tables, forms, business functions, data dictionary items, batch processes, business views, event rules, versions, data structures, and media objects.

EnterpriseOne process A software process that enables JD Edwards EnterpriseOne clients and servers to handle processing requests and run transactions. A client runs one process, and servers can have multiple instances of a process. JD Edwards EnterpriseOne processes can also be dedicated to specific tasks (for example, workflow messages and data replication) to ensure that critical processes don't have to wait if the server is particularly busy.

Environment Workbench An application that, during the Installation Workbench process, copies the environment information and Object Configuration Manager tables for each environment from the Planner data source to the system-release number data source. It also updates the Environment Plan detail record to reflect completion.

escalation monitor A batch process that monitors pending requests or activities and restarts or forwards them to the next step or user after they have been inactive for a specified amount of time.

event rule A logic statement that instructs the system to perform one or more operations based on an activity that can occur in a specific application, such as entering a form or exiting a field.

facility An entity within a business for which you want to track costs. For example, a facility might be a warehouse location, job, project, work center, or branch/plant. A facility is sometimes referred to as a "business unit."

fast path A command prompt that enables the user to move quickly among menus and applications by using specific commands.

file server A server that stores files to be accessed by other computers on the network. Unlike a disk server, which appears to the user as a remote disk drive, a file server is a sophisticated device that not only stores files, but also manages them and maintains order as network users request files and make changes to these files.

final mode The report processing mode of a processing mode of a program that updates or creates data records.

FTP server A server that responds to requests for files via file transfer protocol.

header information Information at the beginning of a table or form. Header information is used to identify or provide control information for the group of records that follows.

interface table See Z table.

integration server A server that facilitates interaction between diverse operating systems and applications across internal and external networked computer systems.

integrity test A process used to supplement a company's internal balancing procedures by locating and reporting balancing problems and data inconsistencies.

interoperability model A method for third-party systems to connect to or access JD Edwards EnterpriseOne.

in-your-face-error In JD Edwards EnterpriseOne, a form-level property which, when enabled, causes the text of application errors to appear on the form.

IServer service This internet server service resides on the web server and is used to speed up delivery of the Java class files from the database to the client.

jargon An alternative data dictionary item description that JD Edwards EnterpriseOne appears based on the product code of the current object.

Java application server A component-based server that resides in the middle-tier of a server-centric architecture. This server provides middleware services for security and state maintenance, along with data access and persistence.

JDBNET A database driver that enables heterogeneous servers to access each other's data.

JDEBASE Database Middleware A JD Edwards EnterpriseOne proprietary database middleware package that provides platform-independent APIs, along with client-to-server access.

JDECallObject An API used by business functions to invoke other business functions.

jde.ini A JD Edwards EnterpriseOne file (or member for iSeries) that provides the runtime settings required for JD Edwards EnterpriseOne initialization. Specific versions of the file or member must reside on every machine running JD Edwards EnterpriseOne. This includes workstations and servers.

JDEIPC Communications programming tools used by server code to regulate access to the same data in multiprocess environments, communicate and coordinate between processes, and create new processes.

jde.log The main diagnostic log file of JD Edwards EnterpriseOne. This file is always located in the root directory on the primary drive and contains status and error messages from the startup and operation of JD Edwards EnterpriseOne.

JDENET A JD Edwards EnterpriseOne proprietary communications middleware package. This package is a peer-to-peer, message-based, socket-based, multiprocess communications middleware solution. It handles client-to-server and server-to-server communications for all JD Edwards EnterpriseOne supported platforms.

Location Workbench An application that, during the Installation Workbench process, copies all locations that are defined in the installation plan from the Location Master table in the Planner data source to the system data source.

logic server A server in a distributed network that provides the business logic for an application program. In a typical configuration, pristine objects are replicated on to the logic server from the central server. The logic server, in conjunction with workstations, actually performs the processing required when JD Edwards EnterpriseOne software runs.

MailMerge Workbench An application that merges Microsoft Word 6.0 (or higher) word-processing documents with JD Edwards EnterpriseOne records to automatically print business documents. You can use MailMerge Workbench to print documents, such as form letters about verification of employment.

master business function (MBF) An interactive master file that serves as a central location for adding, changing, and updating information in a database. Master business functions pass information between data entry forms and the appropriate tables. These master functions provide a common set of functions that contain all of the necessary default and editing rules for related programs. MBFs contain logic that ensures the integrity of adding, updating, and deleting information from databases.

master table See published table.

matching document A document associated with an original document to complete or change a transaction. For example, in JD Edwards EnterpriseOne Financial Management, a receipt is the matching document of an invoice, and a payment is the matching document of a voucher.

media storage object Files that use one of the following naming conventions that are not organized into table format: Gxxx, xxxGT, or GTxxx.

message center A central location for sending and receiving all JD Edwards EnterpriseOne messages (system and user generated), regardless of the originating application or user.

messaging adapter An interoperability model that enables third-party systems to connect to JD Edwards EnterpriseOne to exchange information through the use of messaging queues.

messaging server A server that handles messages that are sent for use by other programs using a messaging API. Messaging servers typically employ a middleware program to perform their functions.

named event rule (NER) Encapsulated, reusable business logic created using event rules, rather that C programming. NERs are also called business function event rules. NERs can be reused in multiple places by multiple programs. This modularity lends itself to streamlining, reusability of code, and less work.

nota fiscal In Brazil, a legal document that must accompany all commercial transactions for tax purposes and that must contain information required by tax regulations.

nota fiscal factura In Brazil, a nota fiscal with invoice information.

See also *nota fiscal.*

Object Configuration Manager (OCM) In JD Edwards EnterpriseOne, the object request broker and control center for the runtime environment. OCM keeps track of the runtime locations for business functions, data, and batch applications. When one of these objects is called, OCM directs access to it using defaults and overrides for a given environment and user.

Object Librarian A repository of all versions, applications, and business functions reusable in building applications. Object Librarian provides check-out and check-in capabilities for developers, and it controls the creation, modification, and use of JD Edwards EnterpriseOne objects. Object Librarian supports multiple environments (such as production and development) and enables objects to be easily moved from one environment to another.

Object Librarian merge A process that blends any modifications to the Object Librarian in a previous release into the Object Librarian in a new release.

Open Data Access (ODA) An interoperability model that enables you to use SQL statements to extract JD Edwards EnterpriseOne data for summarization and report generation.

Output Stream Access (OSA) An interoperability model that enables you to set up an interface for JD Edwards EnterpriseOne to pass data to another software package, such as Microsoft Excel, for processing.

package JD Edwards EnterpriseOne objects are installed to workstations in packages from the deployment server. A package can be compared to a bill of material or kit that indicates the necessary objects for that workstation and where on the deployment server the installation program can find them. It is point-in-time snapshot of the central objects on the deployment server.

package build A software application that facilitates the deployment of software changes and new applications to existing users. Additionally, in JD Edwards EnterpriseOne, a package build can be a compiled version of the software. When you upgrade your version of the ERP software, for example, you are said to take a package build.

Consider the following context: "Also, do not transfer business functions into the production path code until you are ready to deploy, because a global build of business functions done during a package build will automatically include the new functions." The process of creating a package build is often referred to, as it is in this example, simply as "a package build."

package location The directory structure location for the package and its set of replicated objects. This is usually \\deployment server\release\path_code\package\package name. The subdirectories under this path are where the replicated objects for the package are placed. This is also referred to as where the package is built or stored.

Package Workbench An application that, during the Installation Workbench process, transfers the package information tables from the Planner data source to the system-release number data source. It also updates the Package Plan detail record to reflect completion.

planning family A means of grouping end items whose similarity of design and manufacture facilitates being planned in aggregate.

preference profile The ability to define default values for specified fields for a user-defined hierarchy of items, item groups, customers, and customer groups.

print server The interface between a printer and a network that enables network clients to connect to the printer and send their print jobs to it. A print server can be a computer, separate hardware device, or even hardware that resides inside of the printer itself.

pristine environment A JD Edwards EnterpriseOne environment used to test unaltered objects with JD Edwards EnterpriseOne demonstration data or for training classes. You must have this environment so that you can compare pristine objects that you modify.

processing option A data structure that enables users to supply parameters that regulate the running of a batch program or report. For example, you can use processing options to specify default values for certain fields, to determine how information appears or is printed, to specify date ranges, to supply runtime values that regulate program execution, and so on.

production environment A JD Edwards EnterpriseOne environment in which users operate EnterpriseOne software.

production-grade file server A file server that has been quality assurance tested and commercialized and that is usually provided in conjunction with user support services.

program temporary fix (PTF) A representation of changes to JD Edwards EnterpriseOne software that your organization receives on magnetic tapes or disks.

project In JD Edwards EnterpriseOne, a virtual container for objects being developed in Object Management Workbench.

promotion path The designated path for advancing objects or projects in a workflow. The following is the normal promotion cycle (path):

11>21>26>28>38>01

In this path, *11* equals new project pending review, *21* equals programming, *26* equals QA test/review, *28* equals QA test/review complete, *38* equals in production, *01* equals complete. During the normal project promotion cycle, developers check objects out of and into the development path code and then promote them to the prototype path code. The objects are then moved to the productions path code before declaring them complete.

proxy server A server that acts as a barrier between a workstation and the internet so that the enterprise can ensure security, administrative control, and caching service.

published table Also called a master table, this is the central copy to be replicated to other machines. Residing on the publisher machine, the F98DRPUB table identifies all of the published tables and their associated publishers in the enterprise.

publisher The server that is responsible for the published table. The F98DRPUB table identifies all of the published tables and their associated publishers in the enterprise.

pull replication One of the JD Edwards EnterpriseOne methods for replicating data to individual workstations. Such machines are set up as pull subscribers using JD Edwards EnterpriseOne data replication tools. The only time that pull subscribers are notified of changes, updates, and deletions is when they request such information. The request is in the form of a message that is sent, usually at startup, from the pull subscriber to the server machine that stores the F98DRPCN table.

QBE An abbreviation for query by example. In JD Edwards EnterpriseOne, the QBE line is the top line on a detail area that is used for filtering data.

real-time event A service that uses system calls to capture JD Edwards EnterpriseOne transactions as they occur and to provide notification to third-party software, end users, and other JD Edwards EnterpriseOne systems that have requested notification when certain transactions occur.

refresh A function used to modify JD Edwards EnterpriseOne software, or subset of it, such as a table or business data, so that it functions at a new release or cumulative update level, such as B73.2 or B73.2.1.

replication server A server that is responsible for replicating central objects to client machines.

quote order In JD Edwards Procurement and Subcontract Management, a request from a supplier for item and price information from which you can create a purchase order.

	In JD Edwards Sales Order Management, item and price information for a customer who has not yet committed to a sales order.
selection	Found on JD Edwards EnterpriseOne menus, a selection represents functions that you can access from a menu. To make a selection, type the associated number in the Selection field and press Enter.
Server Workbench	An application that, during the Installation Workbench process, copies the server configuration files from the Planner data source to the system-release number data source. It also updates the Server Plan detail record to reflect completion.
spot rate	An exchange rate entered at the transaction level. This rate overrides the exchange rate that is set up between two currencies.
Specification merge	A merge that comprises three merges: Object Librarian merge, Versions List merge, and Central Objects merge. The merges blend customer modifications with data that accompanies a new release.
specification	A complete description of a JD Edwards EnterpriseOne object. Each object has its own specification, or name, which is used to build applications.
Specification Table Merge Workbench	An application that, during the Installation Workbench process, runs the batch applications that update the specification tables.
store-and-forward	The mode of processing that enables users who are disconnected from a server to enter transactions and then later connect to the server to upload those transactions.
subscriber table	Table F98DRSUB, which is stored on the publisher server with the F98DRPUB table and identifies all of the subscriber machines for each published table.
supplemental data	Any type of information that is not maintained in a master file. Supplemental data is usually additional information about employees, applicants, requisitions, and jobs (such as an employee's job skills, degrees, or foreign languages spoken). You can track virtually any type of information that your organization needs. For example, in addition to the data in the standard master tables (the Address Book Master, Customer Master, and Supplier Master tables), you can maintain other kinds of data in separate, generic databases. These generic databases enable a standard approach to entering and maintaining supplemental data across JD Edwards EnterpriseOne systems.
table access management (TAM)	The JD Edwards EnterpriseOne component that handles the storage and retrieval of use-defined data. TAM stores information, such as data dictionary definitions; application and report specifications; event rules; table definitions; business function input parameters and library information; and data structure definitions for running applications, reports, and business functions.
Table Conversion Workbench	An interoperability model that enables the exchange of information between JD Edwards EnterpriseOne and third-party systems using non-JD Edwards EnterpriseOne tables.
table conversion	An interoperability model that enables the exchange of information between JD Edwards EnterpriseOne and third-party systems using non-JD Edwards EnterpriseOne tables.
table event rules	Logic that is attached to database triggers that runs whenever the action specified by the trigger occurs against the table. Although JD Edwards EnterpriseOne enables event rules to be attached to application events, this functionality is application specific. Table event rules provide embedded logic at the table level.
terminal server	A server that enables terminals, microcomputers, and other devices to connect to a network or host computer or to devices attached to that particular computer.

three-tier processing The task of entering, reviewing and approving, and posting batches of transactions in JD Edwards EnterpriseOne.

three-way voucher match In JD Edwards Procurement and Subcontract Management, the process of comparing receipt information to supplier's invoices to create vouchers. In a three-way match, you use the receipt records to create vouchers.

transaction processing (TP) monitor A monitor that controls data transfer between local and remote terminals and the applications that originated them. TP monitors also protect data integrity in the distributed environment and may include programs that validate data and format terminal screens.

transaction set An electronic business transaction (electronic data interchange standard document) made up of segments.

trigger One of several events specific to data dictionary items. You can attach logic to a data dictionary item that the system processes automatically when the event occurs.

triggering event A specific workflow event that requires special action or has defined consequences or resulting actions.

two-way voucher match In JD Edwards Procurement and Subcontract Management, the process of comparing purchase order detail lines to the suppliers' invoices to create vouchers. You do not record receipt information.

User Overrides merge Adds new user override records into a customer's user override table.

variance In JD Edwards Capital Asset Management, the difference between revenue generated by a piece of equipment and costs incurred by the equipment.

In JD Edwards EnterpriseOne Project Costing and JD Edwards EnterpriseOne Manufacturing, the difference between two methods of costing the same item (for example, the difference between the frozen standard cost and the current cost is an engineering variance). Frozen standard costs come from the Cost Components table, and the current costs are calculated using the current bill of material, routing, and overhead rates.

Version List merge The Versions List merge preserves any non-XJDE and non-ZJDE version specifications for objects that are valid in the new release, as well as their processing options data.

visual assist Forms that can be invoked from a control via a trigger to assist the user in determining what data belongs in the control.

vocabulary override An alternate description for a data dictionary item that appears on a specific JD Edwards EnterpriseOne form or report.

wchar_t An internal type of a wide character. It is used for writing portable programs for international markets.

web application server A web server that enables web applications to exchange data with the back-end systems and databases used in eBusiness transactions.

web server A server that sends information as requested by a browser, using the TCP/IP set of protocols. A web server can do more than just coordination of requests from browsers; it can do anything a normal server can do, such as house applications or data. Any computer can be turned into a web server by installing server software and connecting the machine to the internet.

Windows terminal server A multiuser server that enables terminals and minimally configured computers to display Windows applications even if they are not capable of running Windows software themselves. All client processing is performed centrally at the Windows

terminal server and only display, keystroke, and mouse commands are transmitted over the network to the client terminal device.

workbench A program that enables users to access a group of related programs from a single entry point. Typically, the programs that you access from a workbench are used to complete a large business process. For example, you use the JD Edwards EnterpriseOne Payroll Cycle Workbench (P07210) to access all of the programs that the system uses to process payroll, print payments, create payroll reports, create journal entries, and update payroll history. Examples of JD Edwards EnterpriseOne workbenches include Service Management Workbench (P90CD020), Line Scheduling Workbench (P3153), Planning Workbench (P13700), Auditor's Workbench (P09E115), and Payroll Cycle Workbench.

work day calendar In JD Edwards EnterpriseOne Manufacturing, a calendar that is used in planning functions that consecutively lists only working days so that component and work order scheduling can be done based on the actual number of work days available. A work day calendar is sometimes referred to as planning calendar, manufacturing calendar, or shop floor calendar.

workflow The automation of a business process, in whole or in part, during which documents, information, or tasks are passed from one participant to another for action, according to a set of procedural rules.

workgroup server A server that usually contains subsets of data replicated from a master network server. A workgroup server does not perform application or batch processing.

XAPI events A service that uses system calls to capture JD Edwards EnterpriseOne transactions as they occur and then calls third-party software, end users, and other JD Edwards EnterpriseOne systems that have requested notification when the specified transactions occur to return a response.

XML CallObject An interoperability capability that enables you to call business functions.

XML Dispatch An interoperability capability that provides a single point of entry for all XML documents coming into JD Edwards EnterpriseOne for responses.

XML List An interoperability capability that enables you to request and receive JD Edwards EnterpriseOne database information in chunks.

XML Service An interoperability capability that enables you to request events from one JD Edwards EnterpriseOne system and receive a response from another JD Edwards EnterpriseOne system.

XML Transaction An interoperability capability that enables you to use a predefined transaction type to send information to or request information from JD Edwards EnterpriseOne. XML transaction uses interface table functionality.

XML Transaction Service (XTS) Transforms an XML document that is not in the JD Edwards EnterpriseOne format into an XML document that can be processed by JD Edwards EnterpriseOne. XTS then transforms the response back to the request originator XML format.

Z event A service that uses interface table functionality to capture JD Edwards EnterpriseOne transactions and provide notification to third-party software, end users, and other JD Edwards EnterpriseOne systems that have requested to be notified when certain transactions occur.

Z table A working table where non-JD Edwards EnterpriseOne information can be stored and then processed into JD Edwards EnterpriseOne. Z tables also can be used to retrieve JD Edwards EnterpriseOne data. Z tables are also known as interface tables.

Z transaction Third-party data that is properly formatted in interface tables for updating to the JD Edwards EnterpriseOne database.

Index

D

T

U

V

W

X