JD Edwards EnterpriseOne Tools

Connectors Guide

9.2

JD Edwards EnterpriseOne Tools
Connectors Guide

9.2

Part Number: E53577-08

Contents

Preface

Welcome to the JD Edwards EnterpriseOne documentation.

Documentation Accessibility

For information about Oracle's commitment to accessibility, visit the Oracle Accessibility Program website at *http://www.oracle.com/pls/topic/lookup?ctx=acc&id=docacc* .

Access to Oracle Support

Oracle customers that have purchased support have access to electronic support through My Oracle Support. For information, visit *http://www.oracle.com/pls/topic/lookup?ctx=acc&id=info* or visit *http://www.oracle.com/pls/topic/lookup?ctx=acc&id=trs* if you are hearing impaired.

Related Information

For additional information about JD Edwards EnterpriseOne applications, features, content, and training, visit the JD Edwards EnterpriseOne pages on the JD Edwards Resource Library located at:

http://learnjde.com

Conventions

The following text conventions are used in this document:

Convention	Meaning
Bold	Boldface type indicates graphical user interface elements associated with an action or terms defined in text or the glossary.
Italics	Italic type indicates book titles, emphasis, or placeholder variables for which you supply particular values.
`Monospace`	Monospace type indicates commands within a paragraph, URLs, code examples, text that appears on a screen, or text that you enter.
> Oracle by Example	Indicates a link to an Oracle by Example (OBE). OBEs provide hands-on, step- by-step instructions, including screen captures that guide you through a process using your own environment. Access to OBEs requires a valid Oracle account.

1 Introduction to JD Edwards EnterpriseOne Tools Connectors

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 and the 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 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.

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.

After you determine which connector you should use, you must install and configure the connector. Installation and configuration information for COM and Java connectors is provided in this document.

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 the implementation, take advantage of all JD Edwards sources of information, including the installation guides, reference guides, and troubleshooting information.

The following implementation steps need to be performed before working with JD Edwards EnterpriseOne connectors:

1. Install JD Edwards EnterpriseOne and set up a user account.

   See *JD Edwards EnterpriseOne Server Manager Installation Guide*
2. Install JD Edwards EnterpriseOne applications.

   See JD Edwards EnterpriseOne Applications Installation Guide for your platform and database.

2 Understanding 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.

  The chapter, Using COM Connector Events - Guaranteed Events, discusses event notifications and introspection operations.

The COM connector provides a mechanism for running 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.

JD Edwards EnterpriseOne COM Interoperability

This section discusses:

- COM objects
- COM interoperability usage

COM Objects

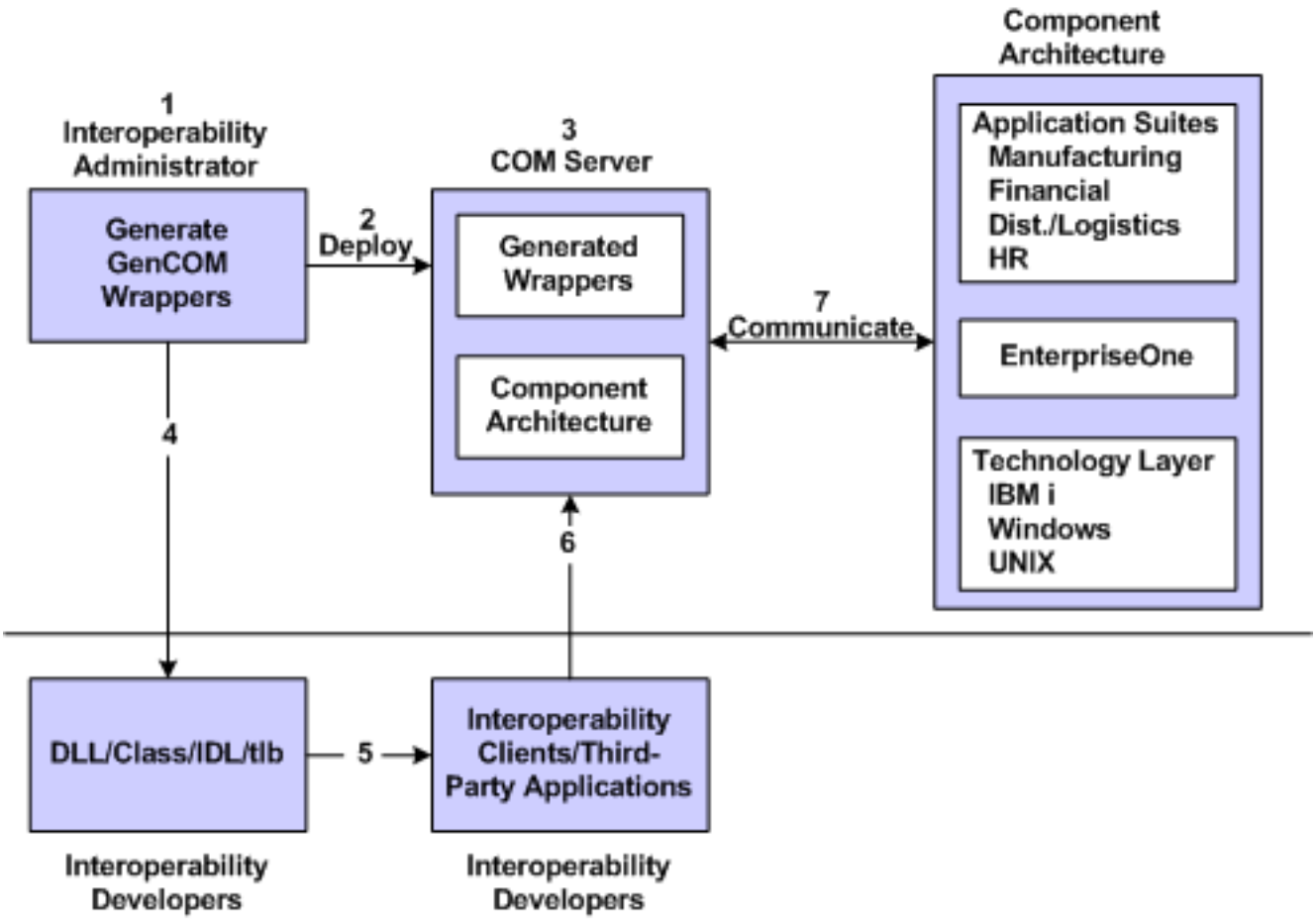
Using COM, JD Edwards EnterpriseOne exposes all master and major business functions through the interface definition language (IDL) standard. A business function is a logical collection of C functions and their associated data structures grouped together to produce a unit of work. With COM, JD Edwards 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
```

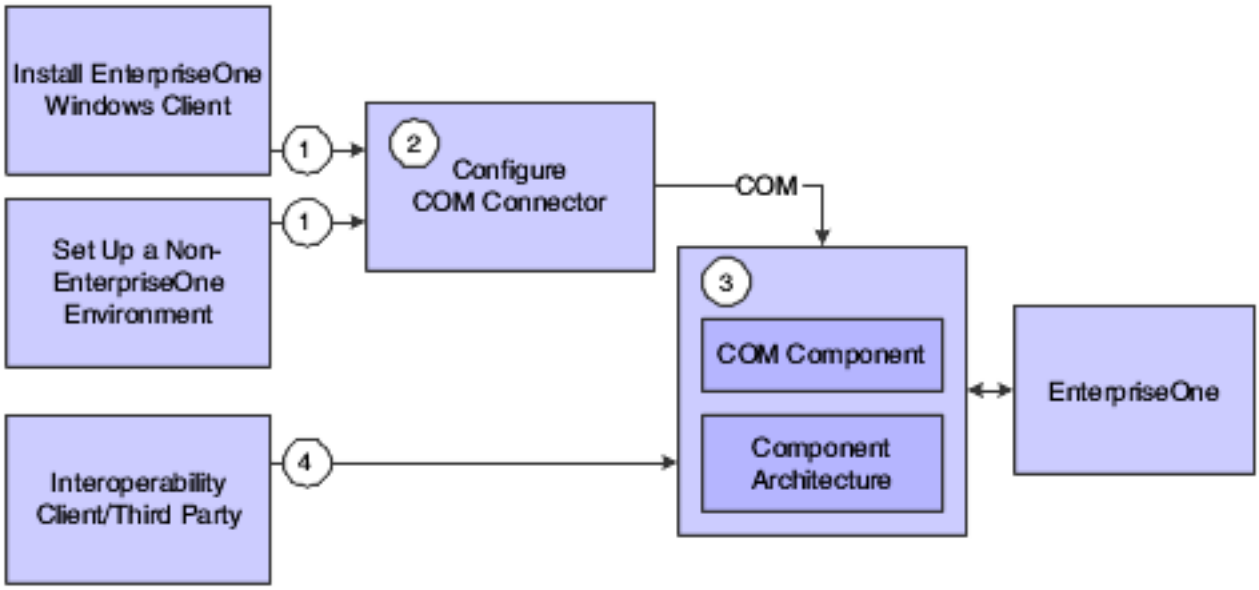
COM Interoperability Usage

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



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:



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 creates applications to subscribe to and receive events.

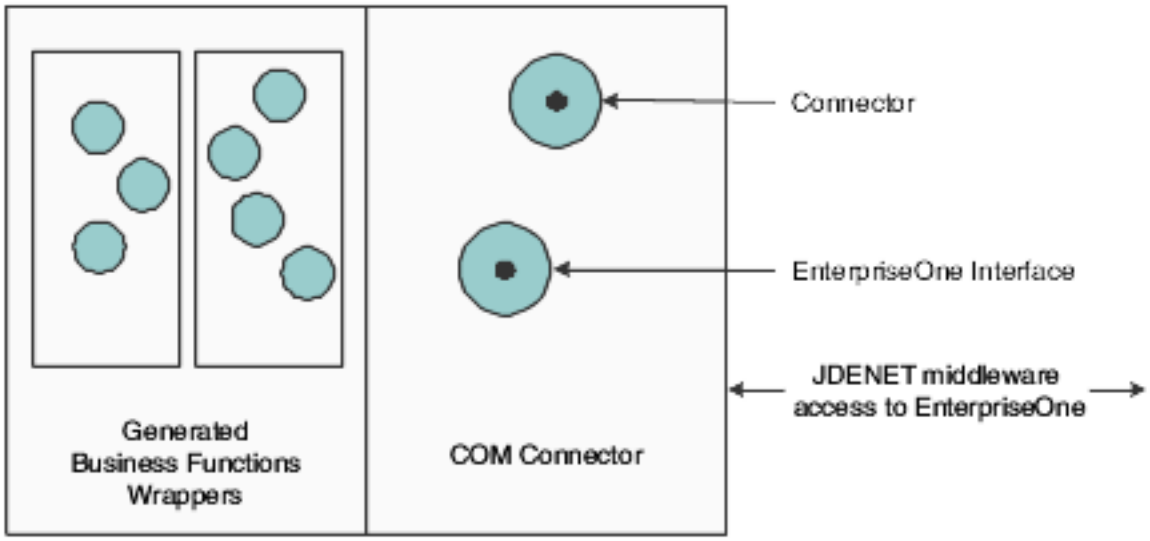
3 Understanding the COM Solution for Business Function Execution

JD Edwards EnterpriseOne COM Server

The JD Edwards EnterpriseOne COM server contains two parts:

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

This diagram shows the two 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.

Binary	Explanation
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+.

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 discusses:

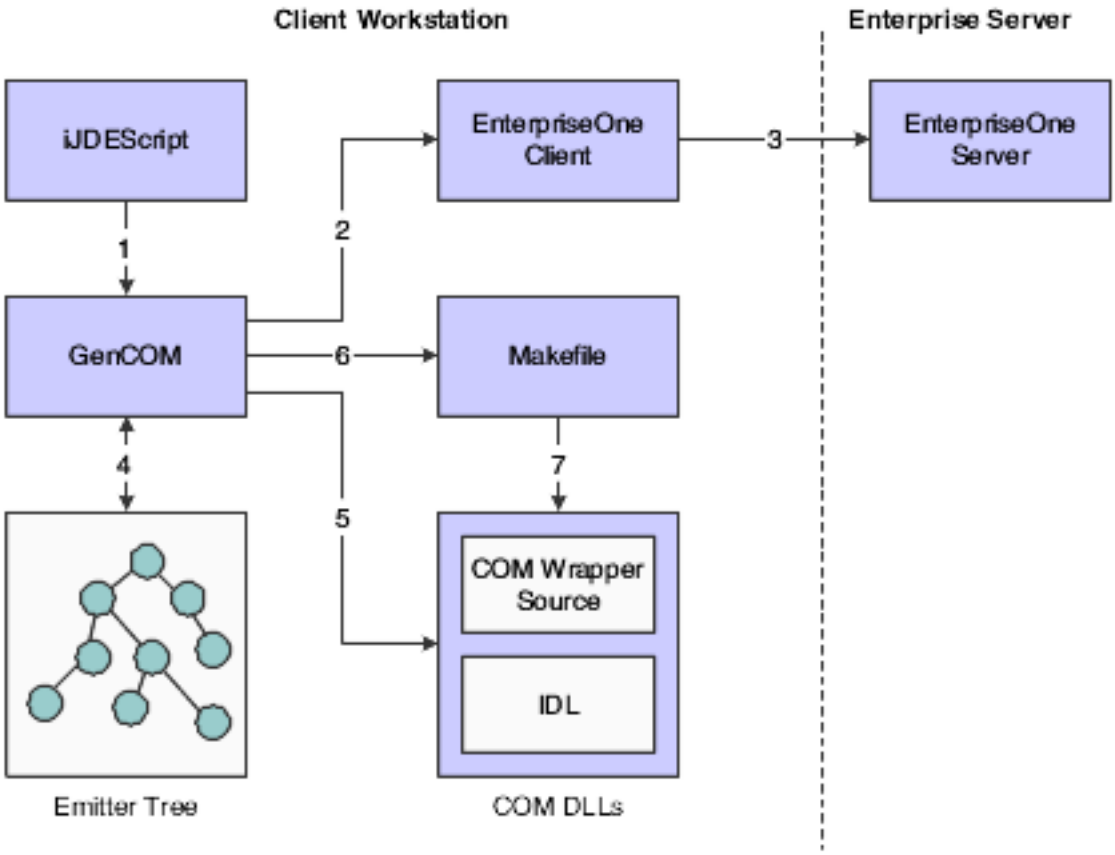
- *Understanding GenCOM*.
- *Installation Information*.
- *ProgID*.
- *Setting Up an Environment for GenCOM*.
- *Running GenCOM*.
- *Using GenCOM Output*.

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:



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 *Understanding iJDEScript*.

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 set up an environment for GenCom on a Microsoft Windows client using Microsoft Visual Studio "X". Setting up the GenCOM environment involves several steps. You should make sure that these items are set up appropriately:

> **Note:** To use Visual Studio "X", your project must have .NET Framework. For example if using Visual Studio 2022, as it supports .NET Framework versions 4.8, 4.7.2, 4.7.1, 4.7, and 4.6.2. Then project must have .NET Framework version for one of those versions.

- Include directories
- Lib Directories
- Paths

Example: Include Directories

Microsoft Visual Studio

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

Below are examples of an include path assuming Visual Studio 2022.

```
C:\Program Files\Microsoft Visual Studio\2022\Professional\VC\Tools\MSVC\14.31.31103\atlmfc\include
```

```
C:\Program Files\Microsoft Visual Studio\2022\Professional\VC\Tools\MSVC\14.31.31103\include
```

JD Edwards EnterpriseOne - Master, Prod, or Pristine

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

Below are examples of an include path for various locations of JD Edwards EnterpriseOne.

```
C:\B9\System\include
```

```
C:\B9\System\includev
```

```
C:\B9\STAGINGA\include
```

Example: Lib Directories

Microsoft Visual Studio

<Directory where Microsoft Visual Studio files are located>**lib

Below are examples of a lib path assuming Visual Studio 2022.

```
C:\Program Files\Microsoft Visual Studio\2022\Professional\VC\Tools\MSVC\14.31.31103\lib\x64
```

```
C:\Program Files\Microsoft Visual Studio\2022\Professional\VC\Tools\MSVC\14.31.31103\atlmfc\lib\x64
```

JD Edwards EnterpriseOne - Master, Prod, or Pristine

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

Below are examples of a lib directory path for various locations of JD Edwards EnterpriseOne.

```
C:\B9\system\Lib32
```

```
C:\B9\system\Libv32
```

```
C:\B9\DV920\Lib32
```

Example: Paths

Microsoft Visual Studio

Below are examples of a path assuming Visual Studio 2022.

```
C:\Program Files\Microsoft Visual Studio\2022\Professional\VC\Tools\MSVC\14.31.31103\bin
```

```
C:\Program Files\Microsoft Visual Studio\2022\Professional\Team Tools\Performance Tools
```

```
C:\Program Files\Microsoft Visual Studio\2022\Professional\\MSBuild\Current\Bin
```

```
C:\Program Files\Microsoft Visual Studio\2022\Professional\Common7\IDE\
```

```
C:\Program Files\Microsoft Visual Studio\2022\Professional\Common7\Tools\
```

Microsoft Windows for use by JD Edwards EnterpriseOne

Below are examples of a path for various locations of JD Edwards EnterpriseOne.

```
C:\E920_1\DV920\bin32

C:\E920_1\system\bin32
```

Running GenCOM

You set up an environment for GenCom on a Microsoft Windows client using a supported version of Microsoft Visual Studio as specified on Oracle Certifications for JD Edwards EnterpriseOne. Your project must use one of the supported .NET Framework versions associated with the Visual Studio Version. For example, if the supported version is Visual Studio 2022, the associated .NET Framework versions would be 4.8, 4.7.2, 4.7.1, 4.7, and 4.6.2.

Setting up the GenCOM environment involves several steps. You should make sure that these items are set up appropriately:

- Include directories
- Lib Directories
- Paths

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

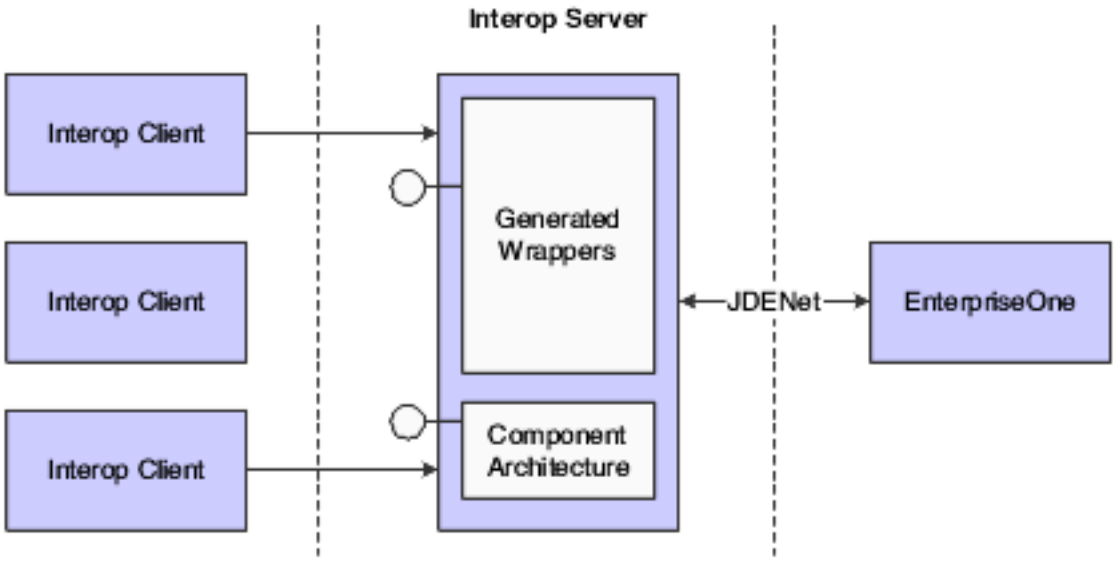
-r -- CheckVer reports only whether the environment is compatible with the server. It does not update the registry settings for the wrapper on the COM server with the result, and CheckVer does not validate the wrapper DLL.

4 Deploying the COM Solution for Business Function Execution

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:



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.
   - CallObject.dll
   - ClientService.dll
   - comlog.dll
   - EventHandler.dll
   - EventListner.dll
   - icui18n.dll
   - icuuc.dll
   - JDECOMConnector2.exe
   - jdecommn.dll
   - jdel.dll
   - jdeunicode.dll
   - PSThread.dll
   - ustdio.dll
   - XERCES4C.dll
   - XercesWrapper.dll
   - XERCESDDOM.dll
   - xmlinterop.dll
   - XMLRequest.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. 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:
   ```
   regsvr32 wrapper.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
```

Note:

- *Understanding jdeinterop.ini for COM Connector*.

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 8.10 (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 *IBM i* , 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.

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 UBE) to the JD Edwards EnterpriseOne server.

See Also

- *"Submit a UBE from XML" in the JD Edwards EnterpriseOne Tools Interoperability Guide .*
- *"Understanding XML CallObject" in the JD Edwards EnterpriseOne Tools Interoperability Guide .*
- "Understanding XML List" in the *JD Edwards EnterpriseOne Tools Interoperability Guide .*

This sample code shows how to use the COM connector to execute an inbound XML request.

```
// File : testDriver.cpp
// Purpose : a test driver to submit the xml request document to OneWorld through
// ThinNet
// Usage : testDriver <input xml doc> <host> <port> <timeout>
// Platform : Win32 Console Program.
// DLL requirement:  xmlinterop.dll, jdeunicode.dll, jdel.dll, jdethread.dll.


#include "iostream"
#include "fstream"
#include "string"
#include   <jde.h>
#include <jdeunicode.h>

extern "C" ZCHAR * JDEWINAPI jdeXMLRequest(const JCHAR *szHostName, unsigned short usPort,
 const int nNetTimeout, void *xml, int size);
extern "C" void JDEWINAPI jdeFreeXMLResponse(ZCHAR *szResp);
```

```
int _cdecl wmain(int argc, wchar_t* argv[], wchar_t* envp[])
{

   ZCHAR     *buf;
   DWORD    dwSize;
   DWORD    dwBytesRead;
   HANDLE   hFile;

   if( argc != 5 )
   {
    std::wcout << _J("Usage: cotest <input xml doc> <host> <port> <timeout>");
      return 0;
   }


  // read the <XML input doc>.
  // Note: the APIs for reading the file are only avaliable in win32.
  if(INVALID_HANDLE_VALUE == (hFile = CreateFile(argv[1], GENERIC_READ,
    0, NULL, OPEN_EXISTING, FILE_ATTRIBUTE_NORMAL, NULL)))
    return 0;

  if(0xFFFFFFFF == (dwSize = ::GetFileSize(hFile, NULL)))
    return 0;

  buf = new ZCHAR[dwSize + 1];
  memset( buf, 0, dwSize+1 );
  if(!ReadFile(hFile, buf, dwSize, &dwBytesRead, NULL))
    return 0;

  CloseHandle(hFile);

   // call C thinNet API to send XML request document
   ZCHAR* presp = jdeXMLRequest(argv[2], jdeAtoi(argv[3]), jdeAtoi(argv[4]), buf, 0);


  // write the XML response into a log file <xmlDoc.log>
  // Note: the APIs for writing the file are only avaliable in win32.
   std::wstring outFile( (JCHAR*)argv[1] );
   std::wstring outExt ( _J(".log") );

   int i;
   if(  (i = outFile.find( _J(".") )) > 0 )
   {
      outFile.replace( i, 4, outExt);

   }
   else
   {
      outFile.append( outExt );
   }
   ZCHAR *outfile = new ZCHAR[jdeStrlen(outFile.c_str())+1];
   jdeFromUnicode(outfile, outFile.c_str(), jdeStrlen(outFile.c_str())+1, NULL);

   std::ofstream outf(outfile);

   outf << presp;

   // free the resource
```

```
   delete [] buf;
   delete[] outfile;
   jdeFreeXMLResponse(presp);


 return 0;
}
```

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.

5 Using COM Transactions

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.
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.

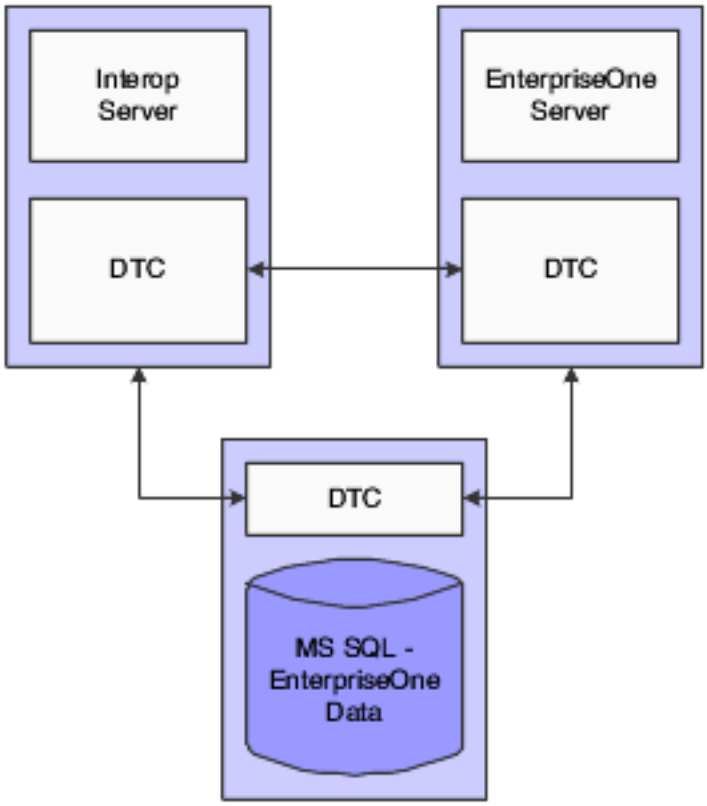
Setting Up the COM+ Environment

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:



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 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.
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.

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.

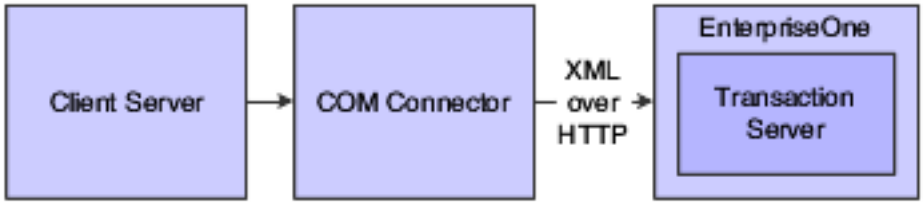
Two cases of the Sales Order Entry application are discussed. Case 1 is when JD Edwards EnterpriseOne participates in the COM+ transaction. Case 2 is when JD Edwards EnterpriseOne participates in a distributed transaction.

6 Using COM Connector Solution for Guaranteed Events

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 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, the COM connector uses XML. This illustration shows the COM connector architecture for guaranteed events:



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.

Note:

- Microsoft MSDN, *http://www.msdn.microsoft.com*.
- *"Using Guaranteed Events" in the JD Edwards EnterpriseOne Tools Interoperability Guide*.
- *"Using Guaranteed Real-Time Events" in the JD Edwards EnterpriseOne Tools Interoperability Guide*.
- *"Using Guaranteed XAPI Events" in the JD Edwards EnterpriseOne Tools Interoperability Guide*.

Setting Up the COM Connector for Guaranteed Events

This section provides an overview of the process for setting up the COM connector to receive guaranteed events.

Understanding COM Connector Setup for Guaranteed Events

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

Use these steps to install and set up the COM connector:

> **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

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.

Note:

- *Installing COM Connector*

Subscribing to Events

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

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:

- *Implementing a JD Edwards EnterpriseOne Interface*
- *Creating a COM+ Component*
- *Logging on to the COM Connector*
- *Subscribing to an Event*
- *Integrating with BizTalk*
- *Adding a New Application*
- *Installing 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 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 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. The name of the application is EventSink, or a name that you prefer. See *Adding a New Application*

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 click 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.

7 Understanding jdeinterop ini File for 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

[OCM]

Configure these [OCM] settings for the COM connector:

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

[JDENET]

Configure these [JDENET] settings for the COM connector:

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

[SERVER]

Configure these [SERVER] settings for the COM connector:

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

[SECURITY]

Configure this [SECURITY] setting for the COM connector.

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

[DEBUG]

Configure these [DEBUG] settings for the COM connector:

Setting and Typical Value	Purpose
JobFile=c:\Interop.log	Location of error file.
DebugFile=c:\InteropDebug.log	Location of debug file.
log=c:\net.log	Location of log file.
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.
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.

[INTEROP]

Configure these [INTEROP] settings for the COM connector:

Setting and Typical Value	Purpose
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.
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.
enterpriseServer=JDED	The JD Edwards EnterpriseOne server.
port=6010	The port number of the JD Edwards EnterpriseOne server.
manual_timout=300000	The time-out value for a transaction in manual commit mode.
Repository=c:\JDEdwards\ Interop \repository	Points to the location of the repository directory containing business object libraries (generated JAR files).

[EVENTS]

Configure these [EVENTS] settings for the COM connector:

Setting and Typical Value	Purpose
UseGuaranteedEvents System= True	Indicates guaranteed event delivery. Values are true and false. Must be set to True to use guaranteed event delivery.
Transport=HTTP	Defines the event transport mechanism. Valid values are HTTP and JMS. The default value is HTTP.
eventServiceURL=http:// <HOST>:<PORT>/ e1events/ EventClientService For a clustered transaction server:	Locates the event service. If the value for the Transport= setting is HTTP, then this setting must be configured. For WebLogic, these ports are the Listen Port. For WebSphere, these ports are the default http ports found under Server > Communication > Ports > WC_defaulthost.

Setting and Typical Value	Purpose
eventServiceURL=http://<HOST1>:<PORT1>/ e1events/ EventClientService|http://<HOST2>:<PORT2>/e1events/ EventClientService If there are more servers in a cluster, then the eventServiceURL can be appended with | as a delimiter; for example: http://<HOST1>:<PORT1>/ e1events/ EventClientService|http://<HOST2>:<PORT2>/ e1events/ EventClientService**|http://<HOST3>:<PORT3>/e1events/ EventClientService**	
jndiProviderURL= **For WebLogic:** jndiProviderURL=t3//<HOST>:<PORT> For a clustered transaction server: t3://<HOST1>:<PORT1>;<HOST2>:<port3> If there are more servers in a cluster, then the jndiProviderURL can be appended with ; as a delimiter; for example: t3://<HOST1>:<PORT1>;<HOST2>:<PORT2>**;<HOST3>;<PORT3>** **For WebSphere:** jndiProviderURL=corbaloc::<HOST>:<PORT NameServiceServerRoot For a clustered transaction server: corbaloc::<HOST1>:<PORT1>, : <HOST2>:<PORT2>/ NameServiceServerRoot If there are more servers in a cluster, then the jndiProviderURL can be appended with ,; as a delimiter; for example: corbaloc::// <HOST1>:<PORT1>;<HOST2>:<PORT2>**;<HOST3>;<PORT3>**/ NameServiceServerRoot	Locates the event service. If the value for the Transport= setting is JMS, then this setting must be configured. For WebLogic, these ports are the Listen Port. for WebSphere, these ports are the Bootstrap ports.
eventReceiveTimeout=60000	Maximum number of milliseconds that the event receiver waits before unsubscribing the event from the JD Edwards EnterpriseOne server.
initialContextFactory= **For WebLogic:**	Initial Context Factory

Setting and Typical Value	Purpose
initialContextFactory=weblogic.jndi.WLInitia **For WebSphere:** initialContextFactory=com.ibm.websphere.r	
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.
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.
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.
Outbound_timeout=1200000	Maximum number of milliseconds that the EventManager waits before unsubscribing the transient event from the JD Edwards EnterpriseOne server.

[JMSEVENTS]

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;EventProcessor_JAR.jar;System_JAR.jar.

Copy the following files from the <JD Edwards EnterpriseOne Windows client installation directory>\system\class folder:

- ApplicationAPIs_JAR.jar
- ApplicationLogic_JAR.jar
- Base_JAR.jar
- BizLogicContainer_JAR.jar
- BizLogicContainerClient_JAR.jar
- BusinessLogicServices_JAR.jar
- castor.jar (Tools Releases prior to 9.2.6)
- commons-codec.jar
- commons-lang2.6.jar
- commons-logging.jar
- Connector.jar
- EventProcessor_JAR.jar
- Generator_JAR.jar

- httpclient.jar
- httpcore.jar
- httpmime.jar
- j2ee1_3.jar
- jakarta.activation.jar (Tools Releases 9.2.6 and greater)
- jakarta.xml.bind-api.jar (Tools Releases 9.2.6 and greater)
- jaxb-core.jar (Tools Releases 9.2.6 and greater)
- jaxb-impl.jar (Tools Releases 9.2.6 and greater)
- JdbjBase_JAR.jar
- JdbjInterfaces_JAR.jar
- JdeNet_JAR.jar
- jmxremote.jar
- jmxremote_optional.jar
- jmxri.jar
- ManagementAgent_JAR.jar
- Metadata.jar
- MetadataInterface.jar
- PMApi_JAR.jar
- Spec_JAR.jar
- System_JAR.jar
- SystemInterfaces_JAR.jar
- xerces.jar
- xml-apis.jar
- xmlparserv2.jar
- The path to the directory where the jdeinterop.ini, jdbj.ini, and jdelog.properties files exist, which must all be in one directory.
- The full path to the JDBC driver files including the filenames.

  The CLASSPATH entry must end with a slash (\), which indicates it is a directory name and not a file name.

Note: For all releases, the files on the client side and Transaction server side must always match. This is important if the Transaction server is updated.

WebSphere

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

- com.ibm.mqjms.jar

  Normally located in the <IBM WebSphere MQ installation directory>/Java/lib folder.

- com.ibm.mq.jar

  Normally located in the <IBM WebSphere MQ installation directory>/Java/lib folder.

- com.ibm.ws.ejb.thinclient_7.0.0.jar

  Normally located in the <WebSphere installation directory>\runtime fold.

- com.ibm.ws.sib.client.thin.jms_7.0.0.jar

  Normally located in the <WebSphere installation directory>\runtime folder.

- com.ibm.ws.orb_7.0.0.jar

  Normally located in the <WebSphere installation directory>\runtime folder.

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

Oracle WebLogic Application Server

If you use WebLogic Application Server for the Java connection, you must include additional files. These files are normally located in the Oracle installation directories in the weblogic.jar folder.

8 Understanding iJDEScript

iJDEScript

GenCOM uses a scripting language called iJDEScript that enables you to script code generation activities. 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.

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 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 enable 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
```

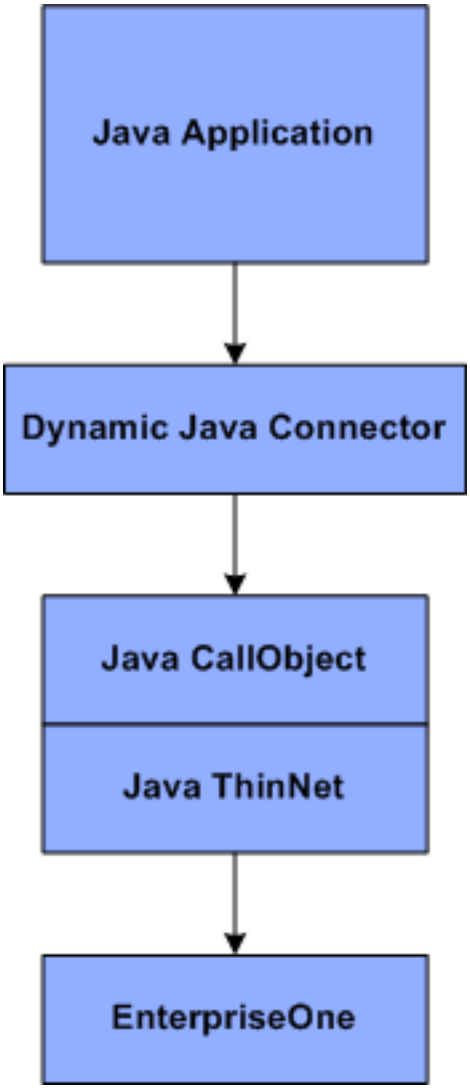
9 Understanding Java Interoperability Solution

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 JD Edwards EnterpriseOne solution uses the dynamic 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 easy for the Java application to switch between JD Edwards EnterpriseOne environments.

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



The dynamic Java 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.

The dynamic Java connector supports the processing of outbound events.

10 Working with the Dynamic Java Connector

Understanding the Dynamic Java Connector

The dynamic Java connector enables a Java application to call a business function. 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.
  Because there is no local storage of business function specification metadata, the business function used by the dynamic Java connector is always compatible with the server specification 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:

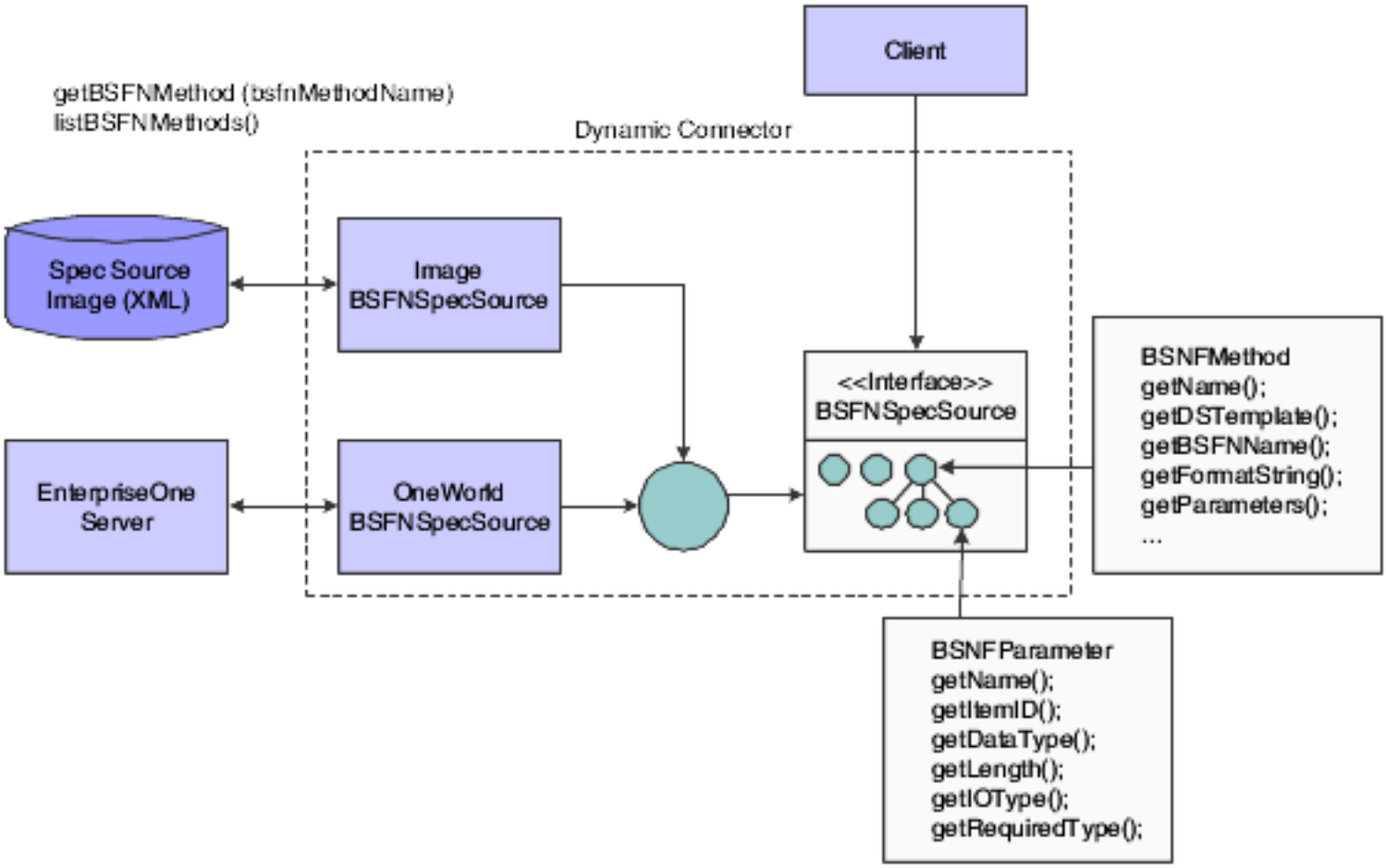
- Public BSFNMethod getBSFNMethod(String methodName) throws SpecFailureException
- Public BSFNMethod[ ] getBSFNMethods() throws SpecFailureException

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 *Installing the Dynamic Java Connector*.

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



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.

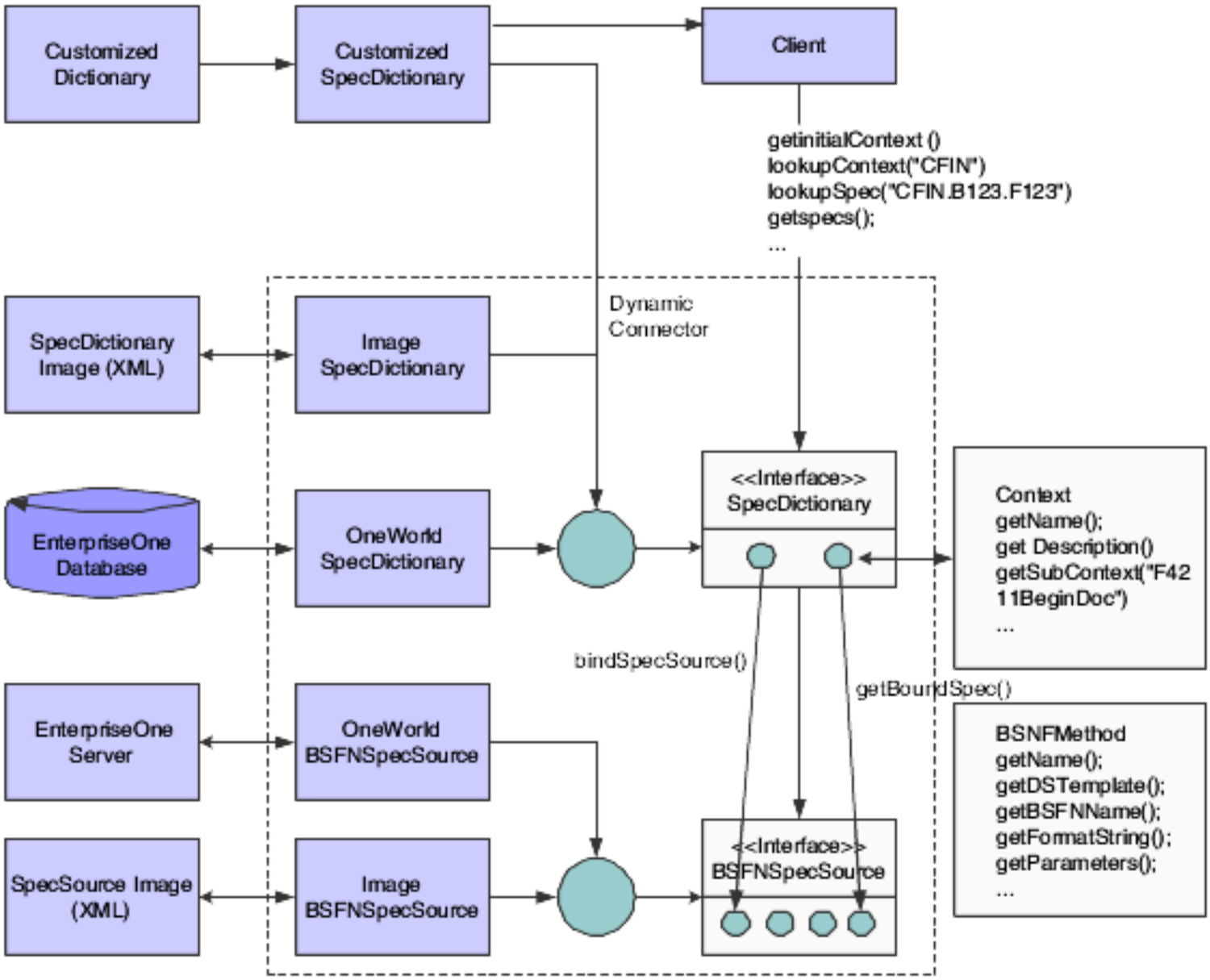
The entry of SpecDictionary is called a context. A context is a set of name-to-object bindings. Every context has an associated naming convention. A context provides a lookup operation that returns the object. The dynamic Java connector provides these two concrete classes that implement the SpecDictionary:

- 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:



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 validate only 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. The SDI and SSI files are generated with the command line code, as illustrated in the following **Example** sections.

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

This section explains how to install dynamic connector components so that you can run a dynamic Java connector application.

Copy the following files from the JD Edwards EnterpriseOne server to a directory on the machine that you want to use (for example, C:\JDEdwards\Interop):

- ApplicationAPIs_JAR.jar
- ApplicationLogic_JAR.jar
- Base_JAR.jar
- BizLogicContainer_JAR.jar
- BizLogicContainerClient_JAR.jar
- BusinessLogicServices_JAR.jar
- castor.jar (Tools Releases prior to 9.2.6)
- commons-codec.jar
- commons-lang2.6.jar
- commons-logging.jar
- Connector.jar
- EventProcessor_JAR.jar
- Generator.jar
- httpclient.jar
- httpcore.jar
- httpmime.jar
- j2ee1_3.jar
- jakarta.activation.jar (Tools Releases 9.2.6 and greater)
- jakarta.xml.bind-api.jar (Tools Releases 9.2.6 and greater)
- jaxb-core.jar (Tools Releases 9.2.6 and greater)
- jaxb-impl.jar (Tools Releases 9.2.6 and greater)
- JdbjBase_JAR.jar
- JdbjInterfaces_JAR.jar
- JdeNet_JAR.jar
- jmxremote.jar
- jmxremote_optional.jar
- jmxri.jar
- ManagementAgent_JAR.jar
- Metadata.jar
- MetadataInterface.jar
- PMApi_JAR.jar
- Spec_JAR.jar
- System_JAR.jar
- SystemInterfaces_JAR.jar
- xerces.jar
- xml-apis.jar
- xmlparserv2.jar

- jdeinterop.ini
- jdbj.ini
- jdelog.properties
- JDBC drivers (obtain the JDBC drivers from the database vendor)

Add all of the copied files to the CLASSPATH.

After you copy the appropriate jar files onto your interoperability machine, do the following:

1. Add the path where the jdelog.properties, jdeinterop.ini, and jdbj.ini files are located into CLASSPATH.
2. 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.

Note:

- *Understanding jdeinterop.ini for Java Connector*.
- *Understanding jdelog.properties File*.

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.

Note:

- *Understanding jdeinterop.ini for Java Connector*.

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.

jdeinterop.ini File Section	Setting	Explanation
[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, XML List, and XML UBE) to the JD Edwards EnterpriseOne server.

See Also

- *"Submit a UBE from XML" in the JD Edwards EnterpriseOne Tools Interoperability Guide .*
- *"Understanding XML CallObject" in the JD Edwards EnterpriseOne Tools Interoperability Guide .*
- *"Understanding XML List" in the JD Edwards EnterpriseOne Tools Interoperability Guide .*

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());
}
    }
```

Logging for the Dynamic Java Connector

Dynamic Java connector logging is built on top java logging. Java logging 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.

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 certified Java Development Kit (JDK). Be sure to install a full JDK and not the Java Runtime Environment (JRE).

  See *Installing the Dynamic Java Connector*.
- 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:**
>
> Minimum technical requirements are updated for each release. See document 745831.1 (JD Edwards EnterpriseOne Minimum Technical Requirements Reference) on My Oracle Support.
> ***https://support.oracle.com/rs?type=doc&id=745831.1***
>
> You can download the JDK from this Oracle website (
> ***http://www.oracle.com/technetwork/java/javase/overview/index.html***
>).

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 the 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.

11 Using Java Connector Guaranteed Events

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:** The terms Java connector and dynamic Java connector refer to the dynamic Java connector. The APIs and the sample code reside in subpackages underneath the com.jdedwards.system.connector.dynamic package. All classes for 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 the dynamic 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.
- Copy jar files to the CLASSPATH.
  - The following jar files must be in the CLASSPATH:
    - ApplicationAPIs_JAR.jar
    - ApplicationLogic_JAR.jar

- Base_JAR.jar
- BizLogicContainer_JAR.jar
- BizLogicContainerClient_JAR.jar
- BusinessLogicServices_JAR.jar
- castor.jar (Tools Releases prior to 9.2.6)
- commons-codec.jar
- commons-lang2.6.jar
- commons-logging.jar
- Connector.jar
- EventProcessor_JAR.jar
- Generator.jar
- httpclient.jar
- httpcore.jar
- httpmime.jar
- j2ee1_3.jar
- jakarta.activation.jar (Tools Releases 9.2.6 and greater)
- jakarta.xml.bind-api.jar (Tools Releases 9.2.6 and greater)
- jaxb-core.jar (Tools Releases 9.2.6 and greater)
- jaxb-impl.jar (Tools Releases 9.2.6 and greater)
- JdbjBase_JAR.jar
- JdbjInterfaces_JAR.jar
- JdeNet_JAR.jar
- jmxremote.jar
- jmxremote_optional.jar
- jmxri.jar
- ManagementAgent_JAR.jar
- Metadata.jar
- MetadataInterface.jar
- PMApi_JAR.jar
- Spec_JAR.jar
- System_JAR.jar
- SystemInterfaces_JAR.jar
- xerces.jar
- xml-apis.jar
- xmlparserv2.jar

The following items are required to compile and run the application or client.

- 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:

  - com.ibm.ws.ejb.thinclient_7.0.0.jar
  - com.ibm.ws.sib.client.thin.jms_7.0.0.jar
  - com.ibm.ws.orb_7.0.0.jar

    > **Note:** These 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. The files that you place on the CLASSPATH must be the exact same files that are on the Transaction server installation directory.

  - com.ibm.ws.orb_7.0.0.jar

    Typically this .jar file is located in the <WebSphere installation directory>/runtime folder.
- If you connect to a Transaction Server hosted on Oracle WebLogic Server, you also need these files:

  - wlclient.jar
  - wljmsclient.jar

    These files can be found in the WebLogic server directory, <WebLogic_Directory>\server\lib.

    > **Note:** Newer versions of the WebLogic server include a new lightweight library called wlthint3client.jar (located at <WebLogic_Directory>\server\lib) that you can use instead of wlclient.jar and wljmsclient.jar.

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 certified Java Runtime Environment (JRE) is installed on the machine.

  You can download a valid JRE from Oracle Technology Network (OTN) web site.

Note:

- *Prerequisites*.
- Java SE Downloads,

  `http://www.oracle.com/technetwork/java/javase/downloads/index.html`
- Document 745831.1 (JD Edwards EnterpriseOne Minimum Technical Requirements Reference) on My Oracle Support.

  `https://support.oracle.com/rs?type=doc&id=745831.1`

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 the 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 the 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 the 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://<HOST>:<PORT>/e1events/EventClient Service For clustered transaction server, specify as:

Section	Setting	Value
		eventServiceURL=http://<HOST1>:<PORT1>/ e1events/ EventClientService|http://<HOST2>: <PORT2>/e1events/ EventClientService For WebLogic, these ports are the Listen port; for WebSphere, these ports are the default http ports found under Server > Communication > Ports > WC_defaulthost. If additional servers are in the cluster, then the eventServiceURL can be appended with | as a delimiter; for example: http://<HOST1>:<PORT1>/e1events/EventClient Service| http://<HOST2>:<PORT2>/e1events/Event ClientService| http://<HOST3>:<PORT3>/e1events/ EventClientService.
[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 the 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 from 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.

12 Understanding jdeinterop.ini for Java Connector

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: When you use Java interoperability connectors, you must also set up jdbj.ini file sections.

Note:

- *JD Edwards EnterpriseOne HTML Server Reference Guide* for your platform.

[OCM]

Configure this [OCM] setting for the dynamic Java connector:

Setting and Typical Value	Purpose
OCMEnabled=True	Selects or clears OCM inside the dynamic Java connector. A value of **true** indicates turned on.

[CACHE]

Configure these [CACHE] settings for the dynamic Java connector:

Setting and Typical Value	Purpose
UserSession=0	Time out value (in milliseconds) for the dynamic Java connector user session. A zero (0) indicates infinite time out.
SpecExpire=1200000	Maximum time (in milliseconds) that the dynamic Java connector keeps the fetched spec in the cache.

[JDENET]

Configure these [JDENET] settings for the Java and dynamic Java connectors:

Setting and Typical Value	Purpose
enterpriseServerTimeout=90000	Timeout value for a request to the JD Edwards EnterpriseOne enterprise server.
maxPoolSize=30	JDENET socket connection pool size.
serviceNameConnect=6004	Port number used by the JD Edwards EnterpriseOne security server.

[SERVER]

Configure these [SERVER] settings for Java and dynamic Java connectors:

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

[SECURITY]

Configure these [SECURITY] settings for Java and dynamic Java connectors:

Setting and Typical Value	Purpose
NumServers=1	Number of security servers set.
SecurityServer=JDED	The JD Edwards EnterpriseOne security server.

[INTEROP]

Configure these INTEROP] settings for Java and dynamic Java connectors:

Setting and Typical Value	Purpose
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.
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.
enterpriseServer=JDED	The JD Edwards EnterpriseOne server.
port=6010	The port number of the JD Edwards EnterpriseOne server.
manual_timout=300000	The time-out value for a transaction in manual commit mode.
Repository=c:\jdedwards\ Interop \repository	Points to the location of the repository directory containing business object libraries (generated JAR files).

[EVENTS]

Configure these [EVENTS] settings for Java and dynamic Java connectors:

Setting and Typical Value	Purpose
UseGuaranteedEvents System=True	Indicates guaranteed event delivery. Values are true and false. Must be set to True to use guaranteed event delivery.
Transport=HTTP	Defines the event transport mechanism. Valued values are HTTP and JMS. The default value is HTTP.
eventServiceURL=http://<HOST>:<PORT>/ e1events/ EventClientService For a clustered transaction server: eventServiceURL=http://<HOST1>:<PORT1>/ e1events/ EventClientService|http://<HOST2>:<PORT2>/e1events/ EventClientService If there are more servers in a cluster, then the eventServiceURL can be appended with | as a delimiter; for example: http://<HOST1>:<PORT1>/ e1events/ EventClientService|http://<HOST2>:<PORT2>/ e1events/ EventClientService**|http://<HOST3>:<PORT3>/e1events/ EventClientService**	Locates the event service. If the value for the Transport= setting is HTTP, then this setting must be configured. For WebLogic, these ports are the Listen Port. For WebSphere, these ports are the default http ports found under Server > Communication > Ports > WC_defaulthost.
jndiProviderURL= **For WebLogic:** jndiProviderURL=t3//<HOST>:<PORT> For a clustered transaction server: t3://<HOST1>:<PORT1>;<HOST2>:<port3> If there are more servers in a cluster, then the jndiProviderURL can be appended with ; as a delimiter; for example: t3://<HOST1>:<PORT1>;<HOST2>:<PORT2>**;<HOST3>;<PORT3>** **For WebSphere:** jndiProviderURL=corbaloc::<HOST>:<PORT NameServiceServerRoot For a clustered transaction server: corbaloc::<HOST1>:<PORT1>, : <HOST2>:<PORT2>/ NameServiceServerRoot If there are more servers in a cluster, then the jndiProviderURL can be appended with ,; as a delimiter; for example:	Locates the event service. If the value for the Transport= setting is JMS, then this setting must be configured. For WebLogic, these ports are the Listen Port. for WebSphere, these ports are the Bootstrap ports.

Setting and Typical Value	Purpose
corbaloc::// <HOST1>:<PORT1>;<HOST2>:<PORT2>; **<HOST3>;<PORT3>**/ NameServiceServerRoot	
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.
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.
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.
Outbound_timeout=1200000	Maximum number of milliseconds that the EventManager waits before unsubscribing the transient event from the JD Edwards EnterpriseOne server.

13 Understanding jdelog.properties File

Settings for the jdelog.properties File

The logging utility in the dynamic Java connector and the Java connector is built on top of java logging. 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.
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.
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, EVENTPROCESSOR.

The Tools Reference and HTML Web Server Reference guides provide information for creating and managing jdelog.properties files.

See *JD Edwards EnterpriseOne Deployment Server Reference Guide* for your platform.

See *JD Edwards EnterpriseOne HTML Server Reference Guide for your platform.*

[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 the 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 the C drive.

[LOG1]

FILE=C:\\ConnectorEventsClient\\log\\jas.log LEVEL=APP FORMAT=APPS
MAXFILESIZE=10MBMAXBACKUPINDEX=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=10MBMBMAXBACKUPINDEX=20 COMPONENT=RUNTIME | INTEROP | JDBJ APPEND=TRUE

Index

G

I

J

L

M

O

P

R

S

T

U

W

X