PeopleSoft

EnterpriseOne Connectors Guide 8.9 PeopleBook

September 2003

EnterpriseOne
Connectors Guide 8.9 PeopleBook
SKU REL9EIC0309

Table of Contents

Connectors

Connectors are point-to-point component-based models that allow third-party applications and ERP to share logic and data. J.D. Edwards connector architecture includes Java and Component Object Model (COM) connectors. The J.D. Edwards connector solutions provide the following:

- 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 allow concurrent users.

J.D. Edwards supports the following connectors:

- Java connector – Java is a portable language, so you can easily tie ERP functionality to Java applications.
- Dynamic Java connector – Provides the same type of functionality as the Java connector, but does not require you to generate business function wrappers.
- COM connector – J.D. Edwards COM connector solution is fully compliant with the Microsoft Component Object Model. You can easily tie ERP functionality to Visual Basic and VC++ applications.

J.D. Edwards 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 listed below:

- You can use XML documents to aggregate business function calls into one object, which reduces network traffic.
- Because XML processing is based on the connector architecture, XML processing is scalable and multiple connections can be opened.
- XML processing supports XML CallObject, XMLList, and XMLTrans.

Choosing the Connector Solution

When you are trying to decide which J.D. Edwards connector best satisfies your needs, you should do the following:

- Identify the logic or data that you want to access in ERP.
- 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, the following guidelines can help you decide which connector to choose:

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

After you decide which connector to use, perform the following tasks:

1. Generate business function wrappers, if necessary. The dynamic Java connector does not require generating business function wrappers.
2. Install the interoperability server (connector).
3. Connect your application to the connector.

Sections in this guide explain how to perform the above three tasks.

See Also

- *World Wide Web Consortium (W3C)* XML home page *http://www.w3.org/XML/* for general information about XML

The following topics in the *Interoperability Guide*:

- *Interoperability Overview* for information about the J.D. Edwards models and capabilities that support these models
- *Business Function Calls* for information about choosing which business function to use
- *XML* for information about XML CallObject, XML Transaction, and XML List features
- *Events* for information about Z, real-time, and XAPI events

COM Interoperability Solution

COM allows 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 language. 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 a mechanism for executing business functions on the enterprise server. You use the GenCOM utility on your Windows client to generate wrappers for objects. The wrappers can be deployed on any non-ERP 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 enterprise server for login, logoff, transactions, and for each business function execution call. The COM connector also supports subscribe and publish functionality for ERP events.

Distributed Component Object Model (DCOM) enables COM objects in a distributed environment.

You can use COM+ transactions, which enables COM applications and third-party applications to take part in distributed transactions.

ERP and COM

Using COM, ERP exposes all master and major business functions through the interface definition language (IDL) standard. With COM, ERP can pass logic and data requests to other applications via COM wrappers. 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.

ERP COM Objects

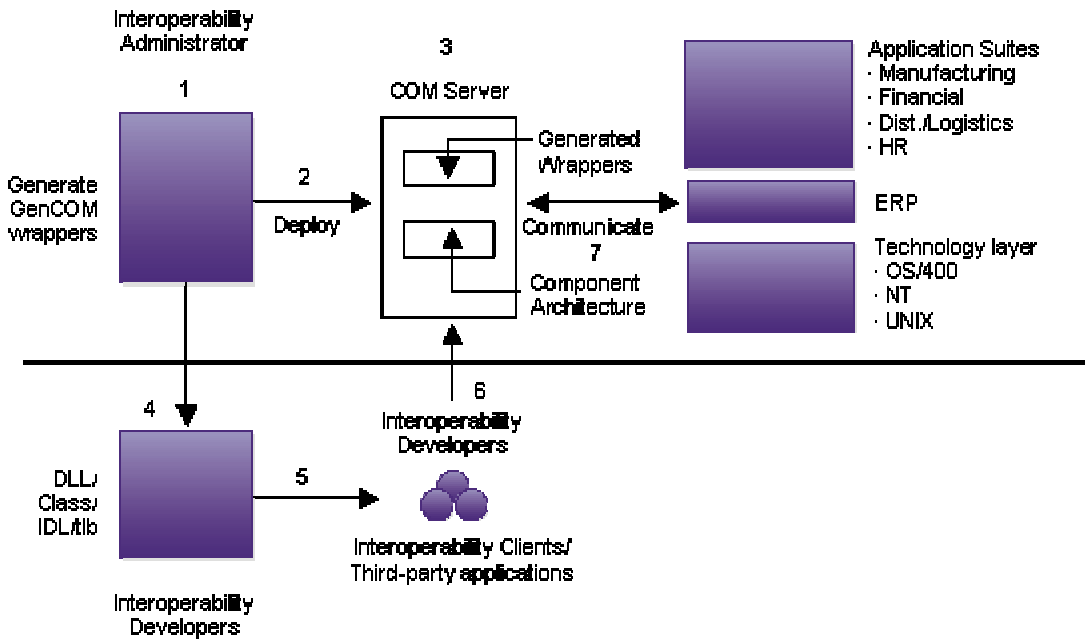
An ERP business function is a logical collection of C functions and their associated data structures grouped together to produce a unit of work. COM objects are wrappers around these business functions and data structures.

The interface provided by the COM wrappers has a one-to-one correspondence with the business functions. For example, if within an ERP 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      - □jdeDS1.jdeDS1.1"
IDS2      - □jdeDS2.jdeDS2.1"
IB550001   - □jdeB550001.jdeB550001.1"
```

COM Interoperability Usage

The following illustration shows how the COM interoperability solution typically flows:



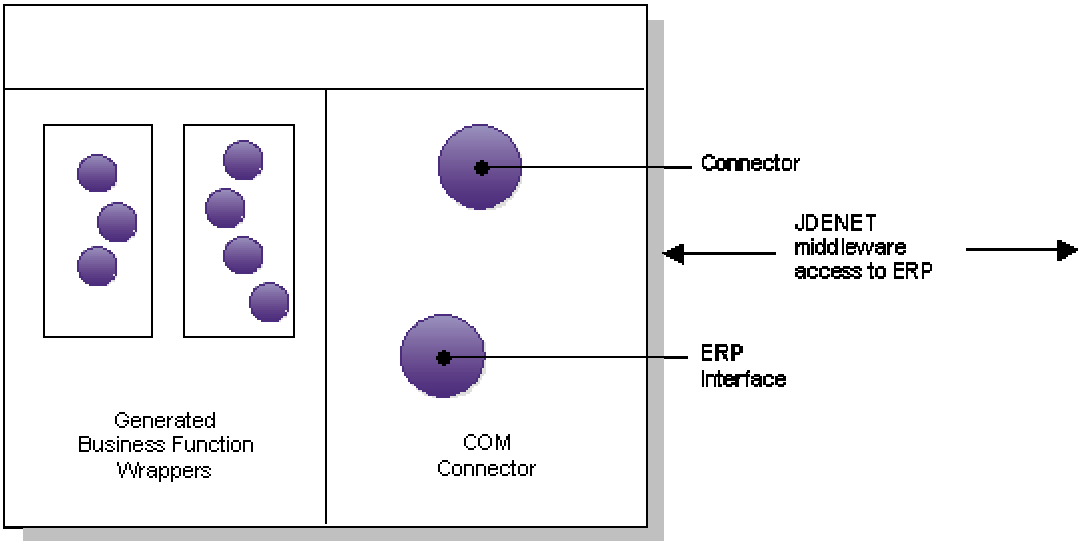
1. [INT: 874974]
2. The administrator generates the COM wrappers.
3. The administrator deploys the COM objects to the COM server.
4. The COM server allows communication with the application server so that the generated COM objects can be used in applications.
5. Once the COM objects are on the COM server, the COM objects are configured to communicate with the application server.
6. The DLLs or IDLs from the generated COM objects are copied so that developers can use them.
7. Application developers create the applications.
8. The applications communicate with the COM server.

ERP COM Server

The ERP COM server contains two parts:

- Generated ERP COM components (wrappers)
- COM connector

The following diagram shows the two parts of the ERP COM server:



COM Connector

The COM server provides an interface to ERP, 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 ERP and hosts the business component DLL generated by the GenCOM tool. The COM connector also provides the connector component that allows an interoperability client to log in and log out from ERP. It manages all user sessions connected to the COM server. The following binaries combine to comprise the COM connector:

Binary	Explanation
JDECOMConnector2.exe	Primary interface for login and createBusinessObjects. Also maintains the users and business objects created.
JDECOMMN.dll	Interface for JDEMathNumeric and JDETimeZone.
Callobject.dll	Internal to JDECOMConnector.exe.
Comlog.dll	Used for logging, cache, and OCM lookup.
EventClass.dll	ERP event class that needs to be implemented to receive events.

EventListener.dll	Receives events from the enterprise 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 JDE.
OneWorldInterfaceTx.dll	Provides the interface for ERP transactions and COM+ two-phase commit transactions.
Xmlinterop.dll	Contains the JDENET transport mechanism and the XMLRequest.

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 ERP enterprise server and client install.

Generated COM Components

GenCOM is included in the client installation. 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.

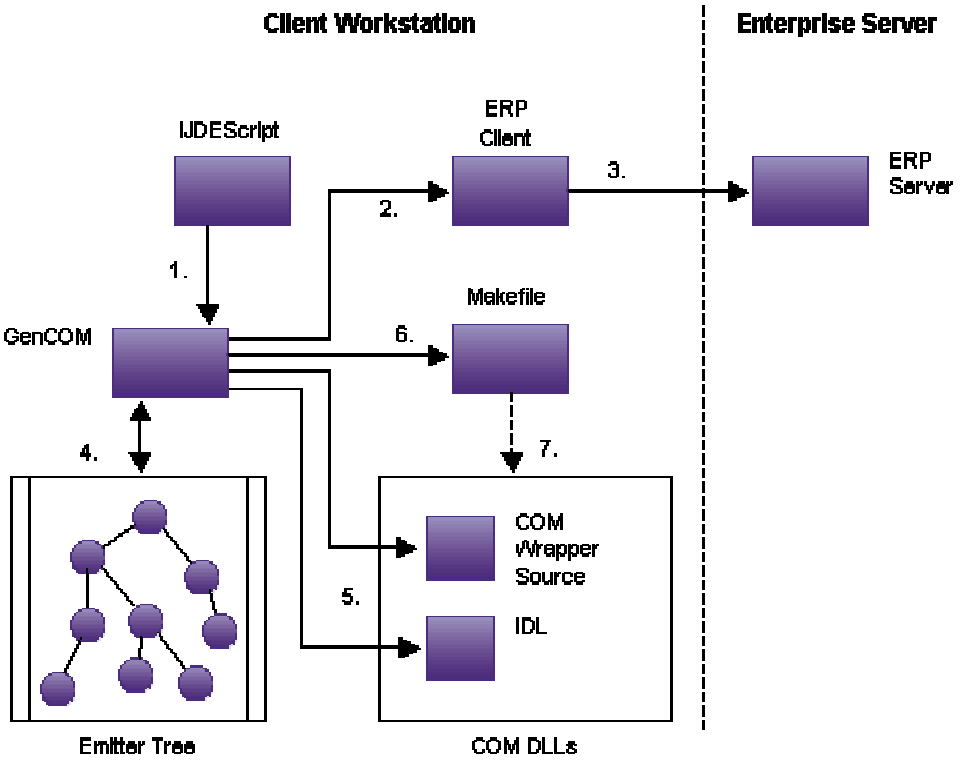
See Also

- *iJDEScript* in the *Connectors Guide* for more information about iJDEScript

GenCOM

GenCOM is a client tool that generates ERP COM components. GenCOM is a command line tool that reads a script file to determine which components to generate. GenCOM uses a multi-pass process to generate COM components.

The following illustration shows the process:



1. [INT: 874976]
2. GenCOM reads the iJDEScript file.
3. GenCOM fetches the metadata for the business functions specified in the iJDEScript file.
4. GenCOM resolves dependency on the data structure.
5. GenCOM creates an internal emitter tree for the library to be generated.
6. GenCOM reads each node of the internal emitter tree and generates the appropriate COM code.
7. GenCOM generates a make file.
8. GenCOM compiles and builds the COM DLL from the generated code.

Setting Up an ERP Environment for GenCOM

Setting up an NT client environment involves several steps. You should make sure that the following items are set up appropriately:

- Include directories
- Lib directories
- MSDev directories
- Paths

Example: Include Directories

< *Directory where Microsoft SDK files are located*>\include

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

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

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

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

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

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

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

< *Directory where PeopleSoft ERP is located and release either Master, Prod, or Pristine*>\include

Example 1: D:\B9\MSTB9\include

Example 2: D:\B9\PROD\include

< *Directory where PeopleSoft ERP is located and release either Master, Prod, or Pristine*>\includeV

Example: D:\B9\SYSTEM\includeV

< *Directory where PeopleSoft ERP is located and release either Master, Prod, or Pristine*>\include

Example: D:\B9\SYSTEM\include

Example: Lib Directories

< *Directory where Microsoft SDK files are located*>\lib

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

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

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

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

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

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

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

< *Directory where PeopleSoft ERP is located*>\System\Lib32

Example: D:\B9\System\Lib32

Example: MSDev Directories

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

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

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

Example: C:\Program Files\DevStudio\SharedIDE

Example: Paths

< *Directory where Microsoft SDK files are located*>\bin

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

< *Directory where Windows NT is located*>\System32

Example: C:\Winnt\System32

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

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

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

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

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

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

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

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

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

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

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

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

< *Directory where PeopleSoft ERP is located*>\System\Bin32

Example: D:\B9\System\Bin32

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

Using the COM Generator (GenCOM)

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

You use the J.D. Edwards scripting language, iJDEScript, to script code generation activities when you use GenCOM.

Running GenCOM

You run GenCOM from the command line. Several options are available for generation. The COM Generation Tool is in <install>\system\bin32\GenCOM.exe.

Syntax

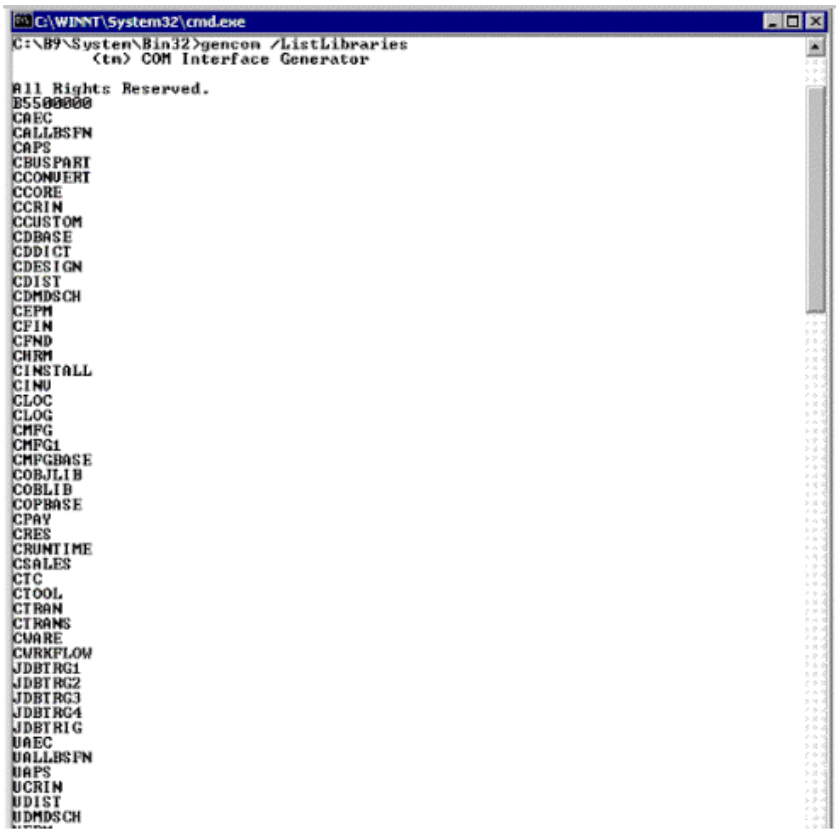
GenCOM [options] [libraries]

Options

/?	Lists the options available for generation.
/C++ <option>	Provides GenCOM with the compiler options you want to use in the generation of the COM servers.
/Cat <category>	Tells GenCOM to generate wrappers based on the following categories: • master business functions • major business functions • minor business functions • uncategorized business functions
/CL <file>	Tells GenCOM what compiler (.exe) to use for compilation.
/Cmd *	Processes code generation commands from the console.
/Cmd <filename>	Processes code generation commands from <filename>.
/Debug	Builds debug information (.pdb and .bsc files) into the libraries so that the Visual Studio debugger can access source information.
/EnvironmentID <env>	Provides GenCOM with the environment in which you want to log in to ERP.
/ErrFile <file>	Provides GenCOM with the filename to log errors produced by GenCOM during the generation process, for example, "errors.log".
/MIDL	Provides GenCOM with the MIDL compiler options you wish to use in the generation of the COM servers.
/MTL <file>	Tells GenCOM which MIDL compiler (.exe) to use for compilation.

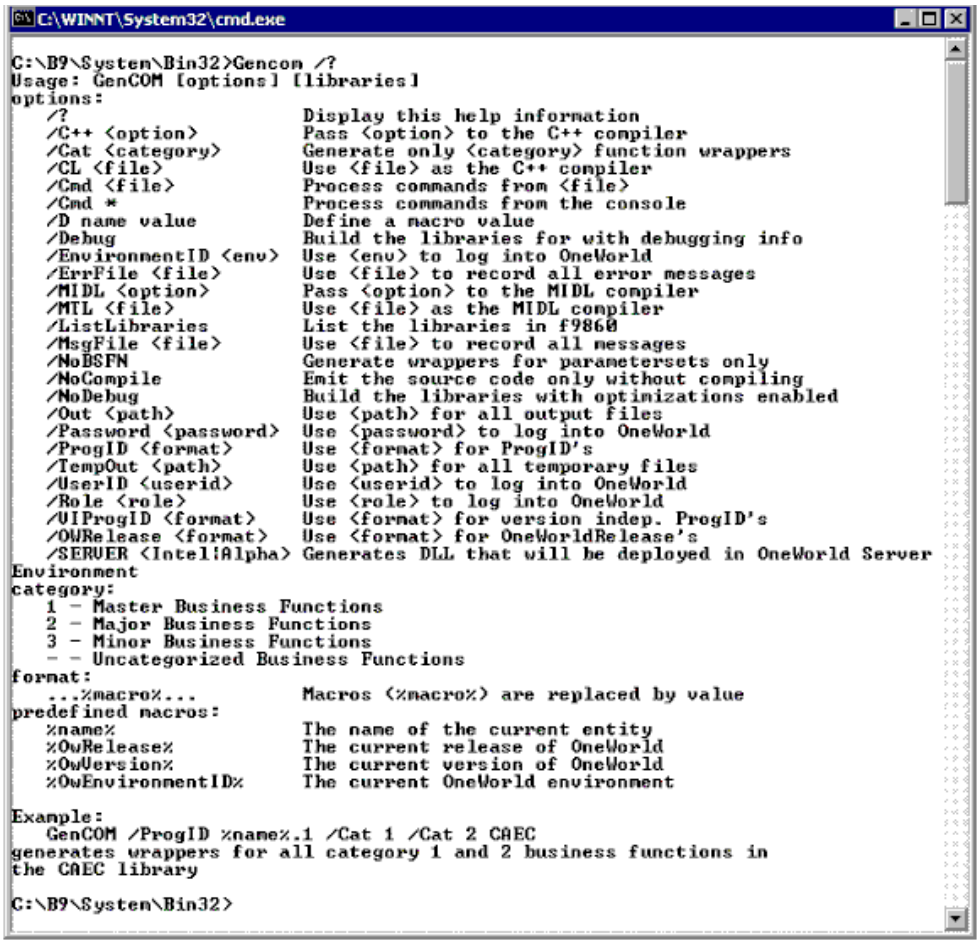
/ListLibraries	Lists all the available libraries that you can run GenCOM against.
/MsgFile <file>	Provides GenCOM with the filename to log messages produced by GenCOM during the generation process, for example, "messages.log"
/NoBSFN	Tells GenCOM not to create wrappers for business functions. This option is for generating parameter sets only.
/NoCompile	Tells GenCOM to only generate the source files without compiling.
/NoDebug	Optimizes libraries for space using the /O1 Visual C++ compiler option.
/Out <path>	Provides GenCOM with the directory path in which to place the output files, for example "C:\winnt\system32".
/OWRelease flag for GenCOM	You can override the OWRelease information by activating this flag and typing a string that specifies the version information. J.D. Edwards recommends that you follow a naming convention that is consistent throughout the implementation or use the default version information that is generated by GenCOM.
/Password <password>	Provides GenCOM with the password with which you want to log in to ERP.
/Role	Provides GenCOM with the role with which you want to log in to ERP.
/STA	Generates STA components. (By default, all generated components are MTA and are optimized for scalability and performance. /STA allows you to generate STA components if you need them.)
/TempOut <path>	Provides GenCOM with the directory path in which to place temporary files needed for the build process, for example, "C:\temp."
/UserID <userid>	Provides GenCOM with the username with which you wish to log in to ERP.

The following is an excerpt from ListLibraries. To see the entire file, run the ListLibraries command from your system.



Example: Running GenCom Entry

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



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 ERP program-independent and is based on the library and the interface specifications in the script file. The key, OneWorldRelease, contains the ERP release and environment information. For example, if the library name is AddressBook and the interface name is JDESalesOrderEntry, then the progID will be AddressBook.JDE AddressBook. If GenCOM is run with environment DV9NIS2, then the OneWorldRelease key will contain DV9NIS2. If there is a type mismatch, you will receive a warning.

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

The following ProgID 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 SalesOrderEntry.JDESalesOrderEntry
\VersionIndependentProgID AddressBook.JDEAddressBook
\OneWorldRelease DV9NIS2
\CompatibleEnvironment  DV9NIS2
```

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 ERP system, you must follow security and installation procedures to gain access to the ERP system.

You must have a fully licensed copy of ERP properly installed on the target machine. You must also log on to the ERP environment. For the logon process, you use the jdeCOMConnector interface.

Visual Basic

The following 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 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()
sessionID=conn.Login("Foo", "Bar", "DV9NIS2")
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++

The following 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"
#inlcude "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");
```

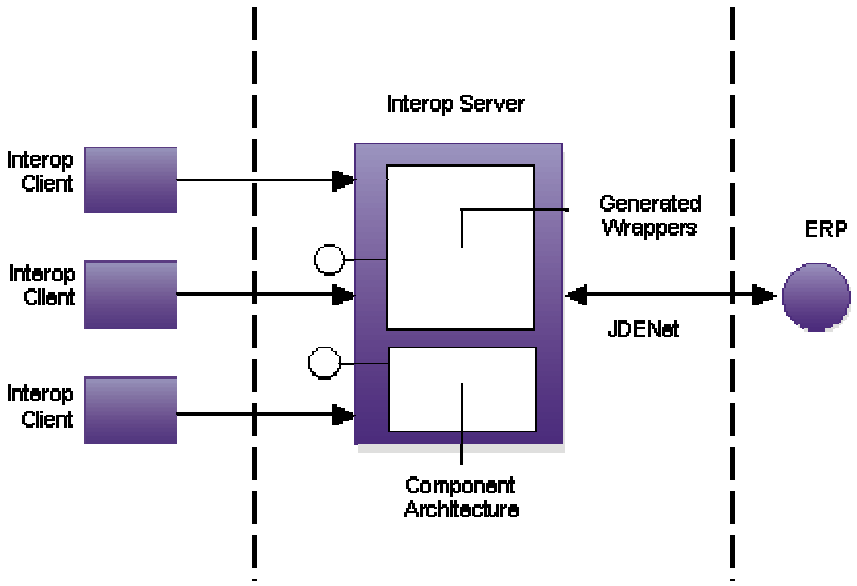
```
  hr = (*pConnector)->Login(User,PassWord,Env,accessno );

  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;
}
```

The following code creates the connector object and uses it to create a J.D. Edwards 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 Server Deployment

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



Installing COM Connector on a Non-ERP Environment

Installing the COM connector on a non-ERP machine requires the following steps:

1. Copy the following files from the enterprise server (system\bin32) to a directory on the desired machine. For example, copy the files in c:\program files\PeopleSoft on to a non-ERP machine.
   - JDECOMConnector2.exe
   - JDECOMMN.dll
   - callobject.dll
   - comlog.dll
   - EventManager.dll
   - OneWorldInterfaceTx.dll
   - xmlinterop.dll
   - jdel.dll
   - jdethread.dll
   - jdeunicode.dll
   - ustdio.dll
   - icuil8n.dll
   - jdeinterop.ini to c:\(root directory)
   - checkver.exe
   - ICUUC.dll
   - Icu\data*.*
   - IXXML4C2_3.dll
   - EventClass.dll
   - EventListener.dll
2. Create a new directory Icu\data\ on the machine where the COM server is located. Copy all of the files from the enterprise 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 the following command on the target location to register the COM connector components:

   c:\programfiles\PeopleSoft\JDECOMConnector2.exe /RegServer

4. Run GenCOM on an ERP client machine and copy the output DLL and the wrapper components (for example, wrapper.dll) to the following directory:

   c:\programfiles\PeopleSoft\wrapper.dll

5. Execute the following command to register the COM wrapper components:

   c:\programfiles\PeopleSoft\regsvr32wrapper.dll

6. Create the JDEinterop.ini file.

   Set the enterpriseServer and port values to the ERP application server with which you want the COM server to communicate.

Your COM server is now ready.

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

```
c:\programfiles\PeopleSoft\JDECOMConnector2.exe /unRegServer
```

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

```
c:\programfiles\PeopleSoft\regsvr32 /u wrapper.dll
```

See Also

- *jdeinterop.ini* in the *Connectors Guide* for more information about the JDEinterop.ini file

Setting Up a DCOM Server

You can set up a DCOM server on an ERP enterprise server machine. DCOM enables COM objects in a distributed environment. The following list identifies the tasks for setting up the DCOM server.

- 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.

To ensure that the interoperability client works properly, you must perform the set up DCOM for a server environment and set up DCOM for a client environment tasks.

► To set up DCOM for a server environment

1. Run GenCOM on an ERP client machine, with the following options:

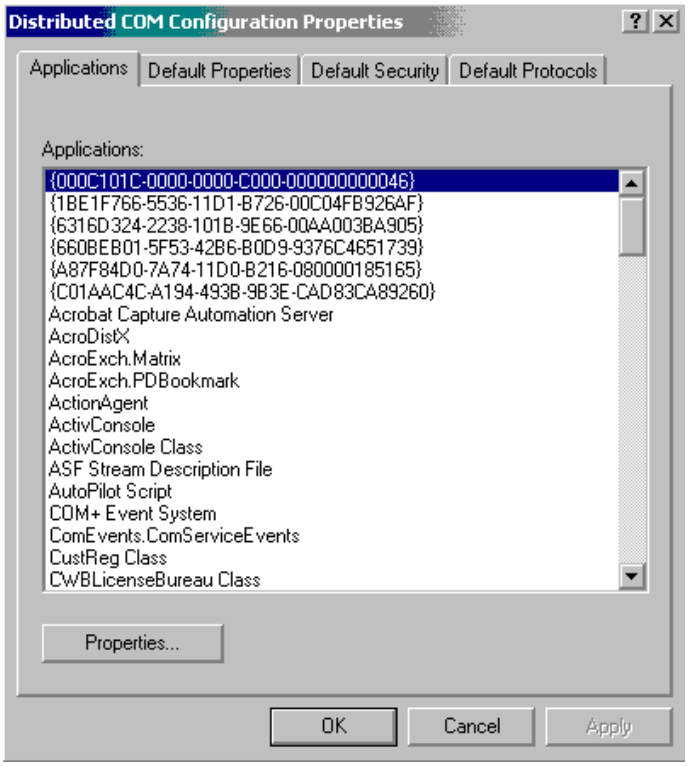
   gencom /out <path> /tempout <path> /cmd App.cmd

   GenCOM is an ERP client side only tool. That is why you must perform this step on an ERP 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

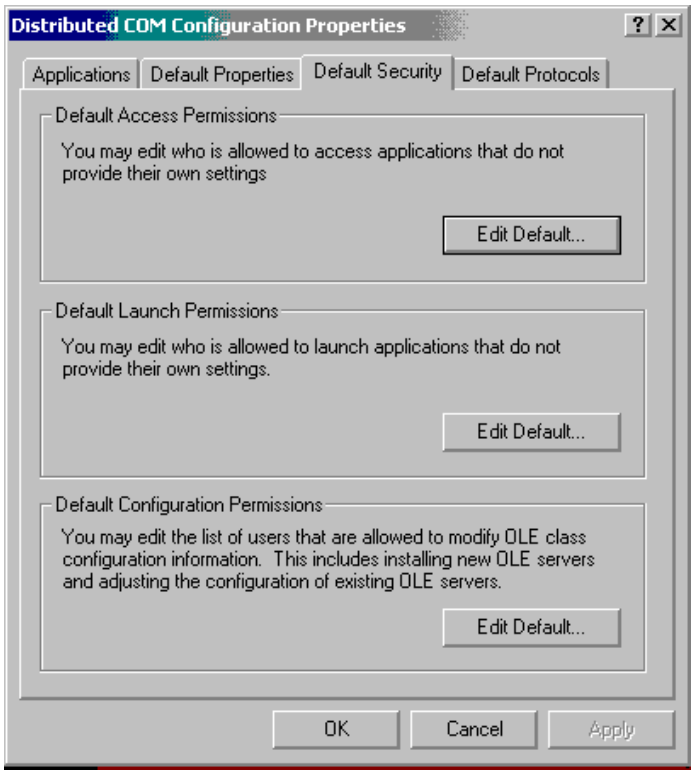
► To set up security on the COM server

1. From the Start menu, choose Run.
2. Enter Dcomcnfg.exe in the Open box.

   The following form appears.

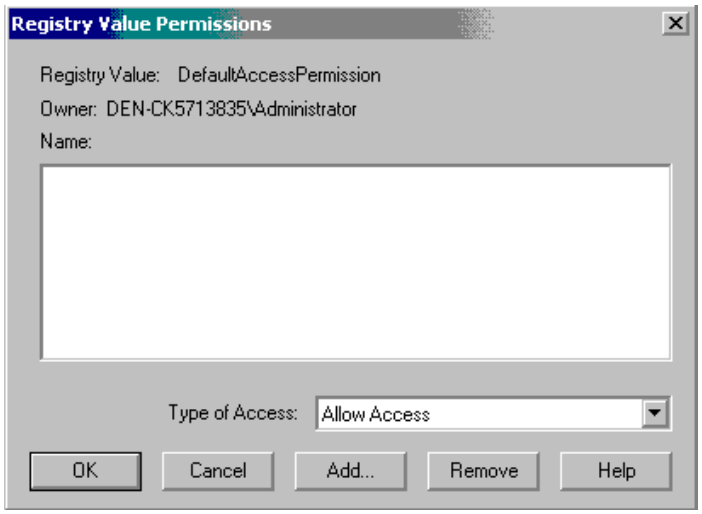


3. Click the Default Security tab.

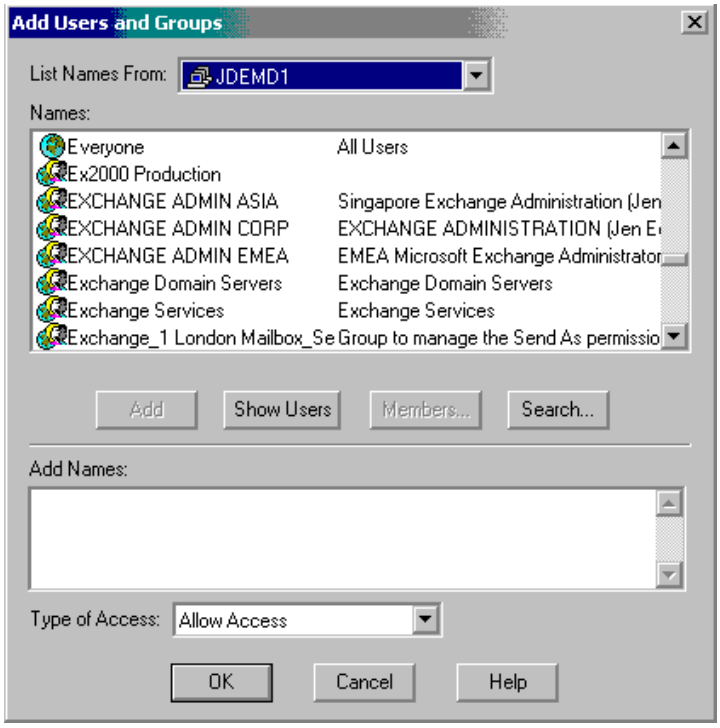


4. Click on the Edit Default Button in Default Access Permissions group.

   The Registry Value Permissions form appears. Your computer might already have some entries present.



5. On Registry Value Permissions, click Add.



6. Click Everyone and click Add.

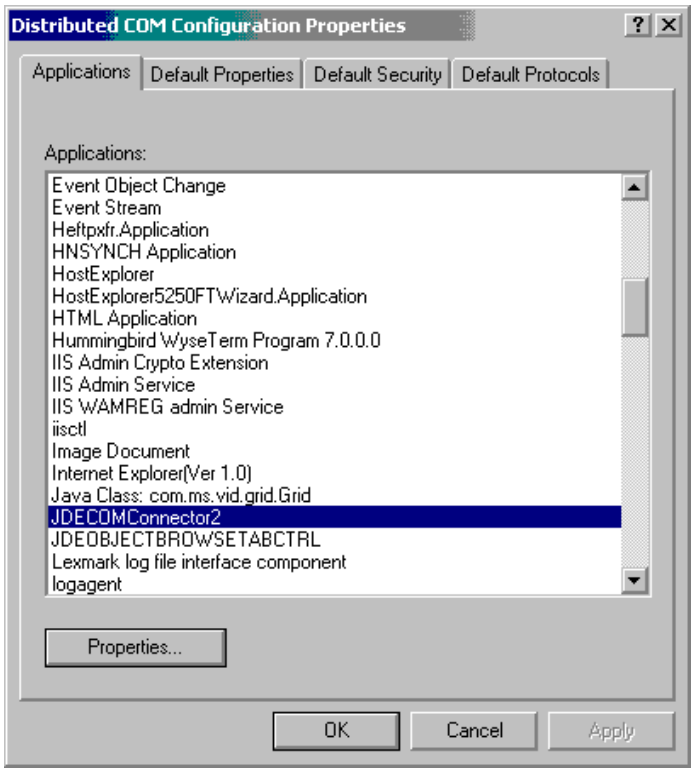
   Type of Access should be Allow Access.

7. Click OK.

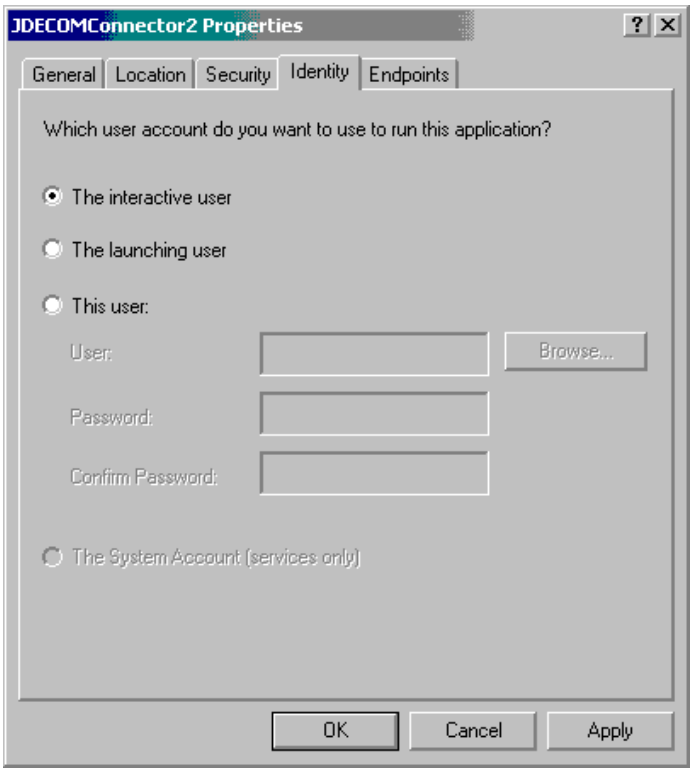
   Repeat Steps 4 through 7 for Default Launch permissions. No setup is required for Default Configuration permissions.

► To set up the identity as interactive user

1. Run DCOMCnfg from the Open box in the Run window.



2. Click JDECOMConnector2 and click Properties.



3. Click the Identity tab and click the interactive user option.
4. Click Apply to apply the change.

Note

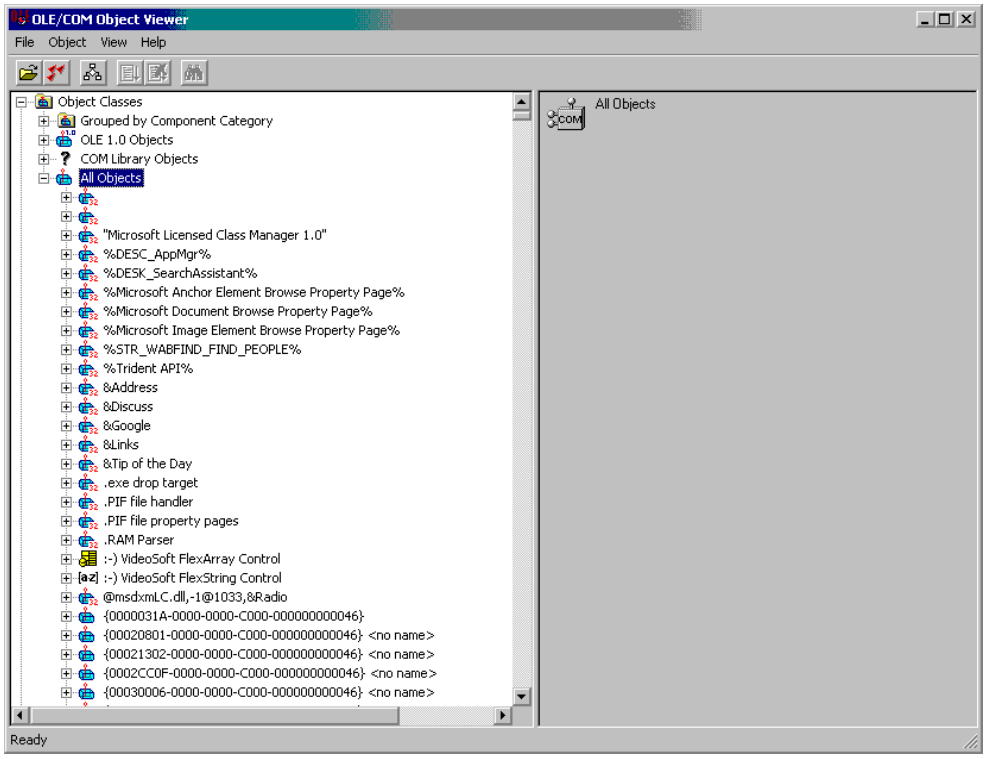
You must perform this step every time you register the connector. If you copy the JDECOMConnector2.exe using Explorer, Explorer will rerun the registration, and you must repeat the above steps.

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.

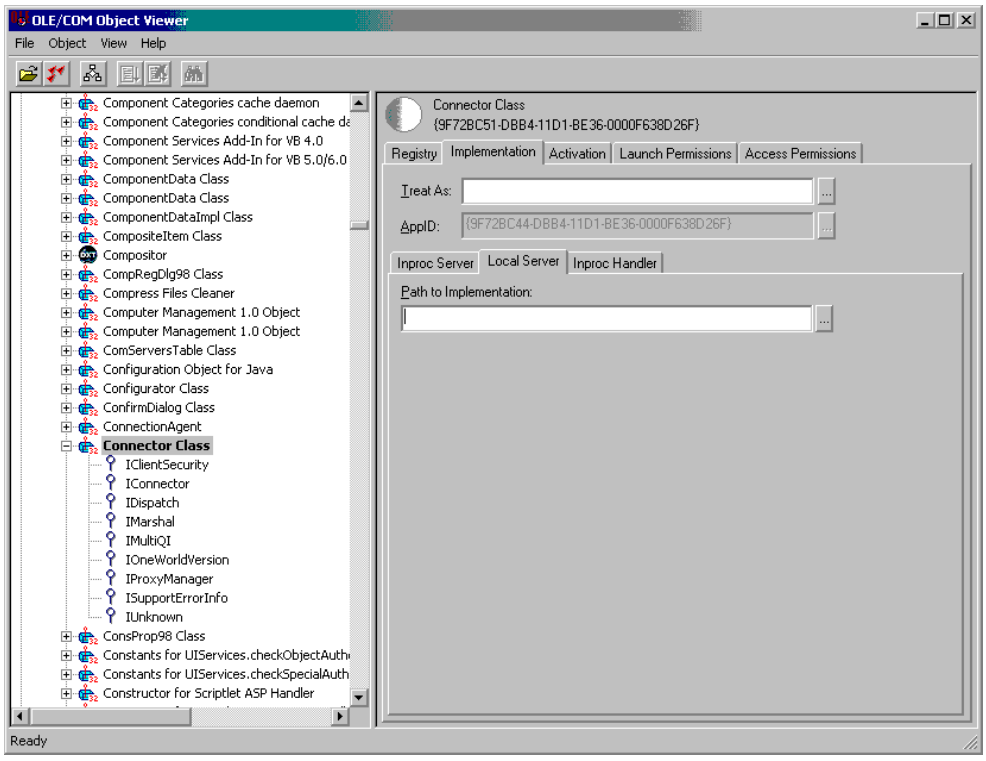
► To set up DCOM for a client environment

1. On the DCOM client machine run jdecomconnector2.exe /RegServer.
2. Enter the following at the prompt:

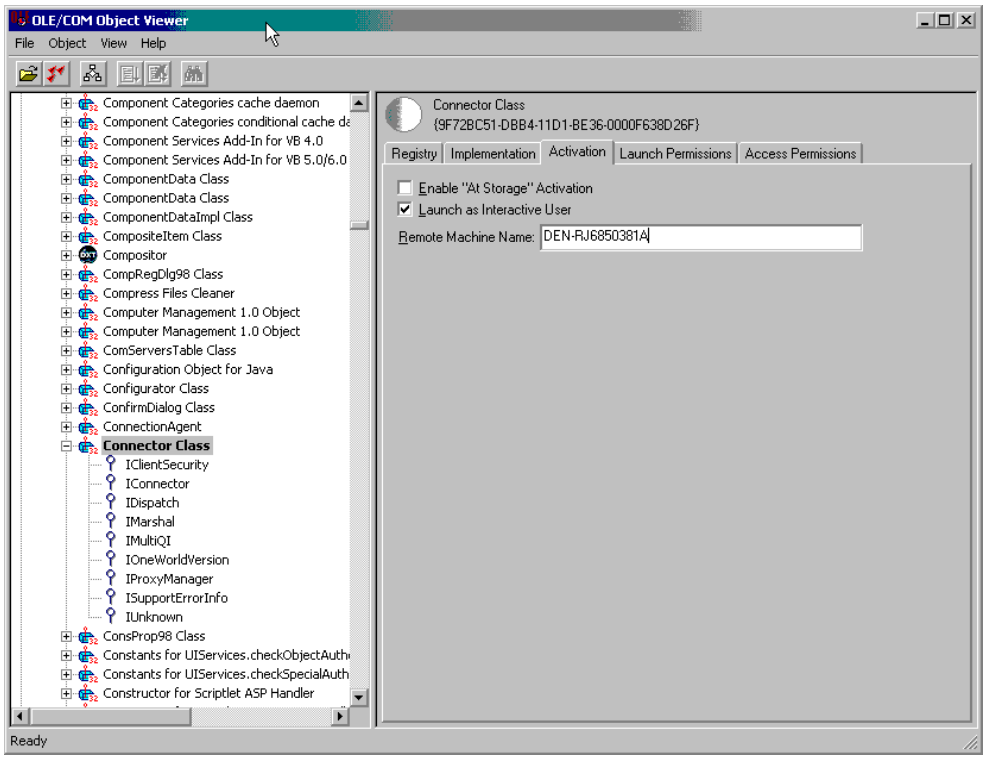
   oleview.exe
3. From the menu bar, choose oleview.
4. Click View and choose Expert Mode.
5. In the oleview window under Object Classes, double-click All Objects, and wait for all objects to appear.



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, turn on 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.

Remember

Client-only ERP business functions are not reachable.

Using the COM Wrapper Version Checker (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 ERP 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 ERP 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 is not allowed to 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 ERP 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 the following command from the command line:

```
c:\>CheckVer.exe -?
```

Syntax

CheckVer [option] <filename>

Example

CheckVer –r addressbook.xml

Options

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

Using the 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 your personal profile or a time zone specified by an application.

The following steps along with sample code illustrate how to use the IJDETimeZone Interface.

1. 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);
    }.
```

2. Set the time for a time zone (UTC-5:30) for the data structure DXXXXXX. If a time zone is not specified, then the time will be considered to be at UTC. If an invalid time zone string is passed, then a error occurs.

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

3. Get a time for a given time zone from ERP. If a time zone string is not passed, then the time and date stored in ERP, which is at UTC, will be 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 an XML files, as follows:

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

Using BHVRCOM via COM

ERP 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.

The following 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
l = conn.Login("JDE", "JDE", "M7332RSO2")
Set OW = conn.CreateBusinessObject("OneWorld.FunctionHelper.1", l)
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", l)
Set phone = AB.CreateGetPhoneParameterset
phone.mnAddressNumber = 1
AB.GetPhone phone, OW, conn, l   `// business function is executed with the BHVRCOM values
... ...
```

Following is an explanation of the some of the above 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.
MyBHYVRCOM.szVersion=	The value is the version of the interactive application. This field can be used for localizations of the applications.

Inbound XML Request via COM Server

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

See Also

See the following topics in the *Interoperability Guide*:

- *XML CallObject* for more information about using XML CallObject
- *XML List* for more information about using XML List
- *XML Transaction* for more information about using XML Transaction

COM Interoperability Transactions

COM interoperability transactions include COM connector prepare, commit, and rollback functionality. The J.D. Edwards COM transaction interoperability solution supports the following types of transactions:

- Auto commit transactions
- Manual commit transactions

In ERP, a transaction can be started as auto commit or manual commit. In auto commit, ERP 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 ERP 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 non-ERP transactions.

Outline for Calling Prepare and Commit

`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 ERP BSFNs`	After a call to Begin Transaction is made, do all the ERP 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.

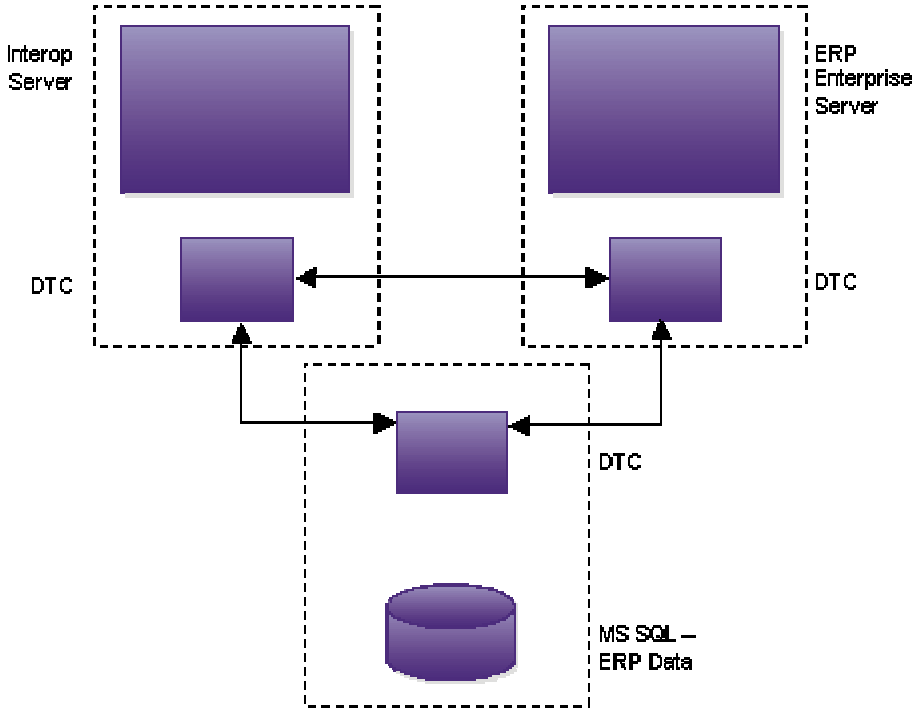
Setting Up the COM+ Environment

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

- COM connector
- Enterprise 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 of the machines is correctly configured and that the DTCs talk to each other.

The following 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 the COM+ Two-Phase Commit Sales Order Entry

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 the following steps to register OneWorldInterfaceTx.dll:

On your PC, navigate to COM+ Applications:

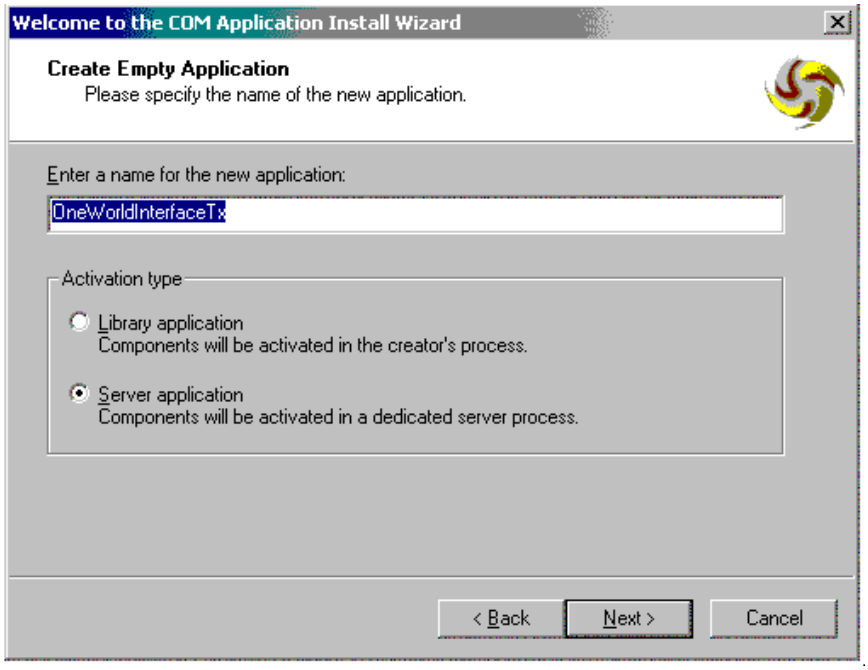
Control Panel > Administrative Tools > Component Services

Expand the following icons and folders:

Component Services > Computers > My Computer

1. Choose COM+ Applications.
2. Right-click COM+ Applications, choose New, and then choose Application.

   The COM Application Install Wizard appears.
3. On Install or Create a New Application, choose Create an empty application and then click Next.

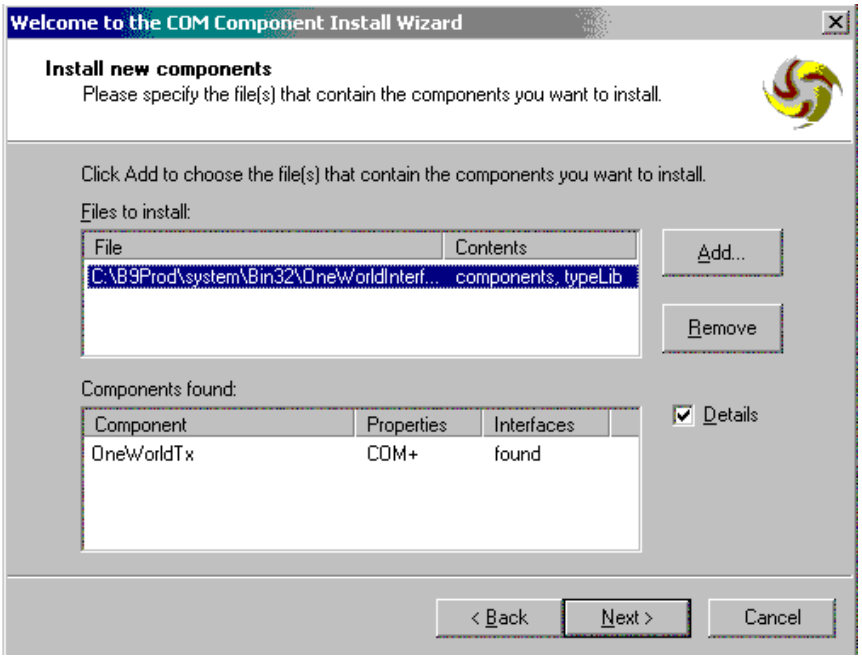


4. On Create Empty Application, enter the name of the application (OneWorldInterfaceTx) that you are registering.
5. Choose an Activation type, and then press Next.
6. On Set Application Identity, choose Interactive User, and then click Next.
7. Click Finish to close the wizard.
8. On your PC, expand the following folders:

   COM+ Applications > OneWorldInterfaceTx.

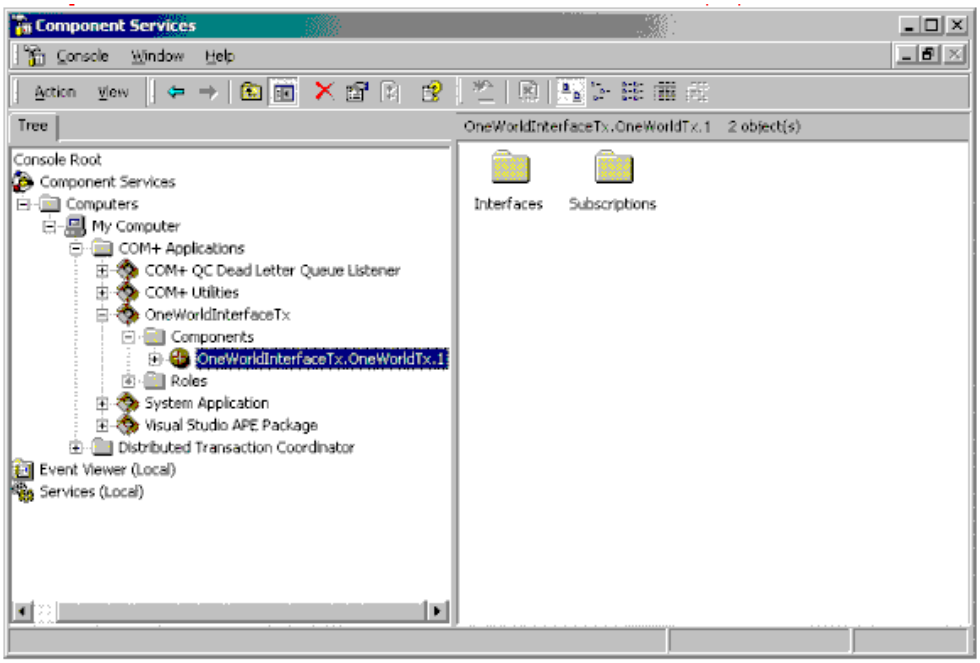
9. Choose Components.
10. Right-click Components, choose New, and then choose Component.

    The COM Component Install Wizard appears.
11. On Import or Install a Component, choose Install New Component(s), and then click Next.
12. On Select New Files to Install, browse to the application (OneWorldInterfaceTx.dll) on the ERP client install directory or the COM interoperability server.

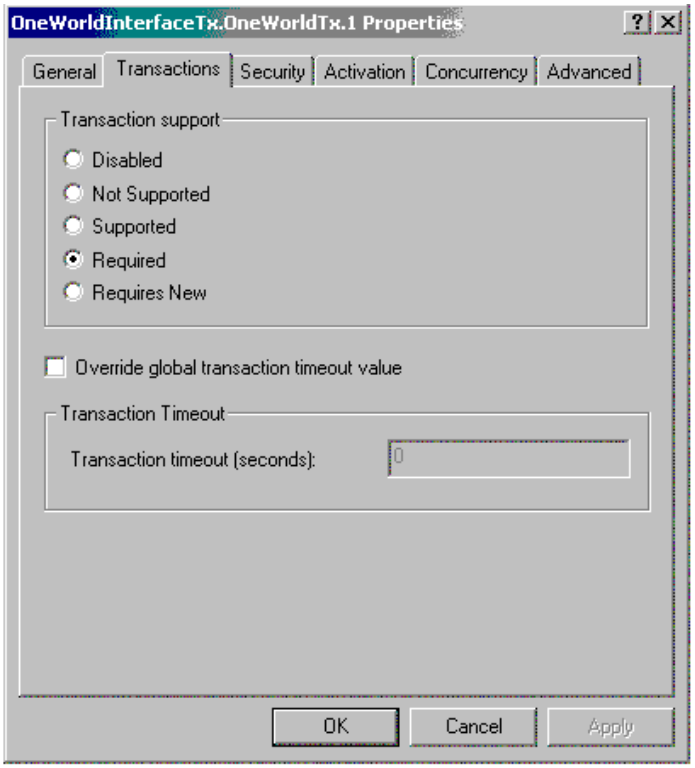


13. Add the application and then click Next.
14. Click Finish to close the wizard.

    Your application (OneWorldInterfaceTx.dll) is registered.



15. On your PC, expand the Components folder and then right-click the application (OneWorldInterfaceTx.dll) you just registered.
16. Choose Properties.



17. On OneWorldInterfaceTx Properties, click the Transactions tab.

18. For the Transaction support field, choose the Required option.
19. Click OK
20. Close the component servers.

The COM connector should be registered using the method described in the chapter titled Installing a COM Connector on a Non-ERP Machine.

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 (follow the steps above). 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: ERP Participation 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 ERP participates in the COM+ transaction. Case 2 is when ERP participates in a distributed transaction.

Case 1: ERP Participates in COM+ Transaction

The following code outline explains how to develop code for COM connector and ERP 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().SetComplet`
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.

The following code examples show you 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 the following steps to run a sales order entry transaction in COM+ where the COM connector and ERP 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, choose the appropriate action.
5. Verify in ERP whether the sales order has been entered. The sales order should be entered only when committed.

Example Code: Sales Order Entry Transactional Object (SOETxObject)

The following sample code shows how to create a SalesOrderEntry transactional object.

```
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", "*ALL")

    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
```

Example Code: Sales Order Entry Transactional Client (SOETxClient)

The following sample code shows how to create a SalesOrderEntry transactional client.

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

Case 2: ERP Participates in a Distributed Transaction

Case 2 explains how a Sales Order Entry and test SQL database query is done as a distributed transaction using COM+.

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

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

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

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

Create a sample SQL test database table SOE2PCTest. SOE2PCTest table will have two columns, SONum and LastSONum. The test chooses 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 the following steps to run a sales order entry as a distributed transaction in COM+ where the COM connector, ERP, 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, choose the appropriate action.
6. Verify in ERP whether the sales order has been entered. When the transaction is aborted, the sales order should not be in ERP, and the test database should not increment the count.

Example Code: MTStest for Distributed Transaction

The following code sample provides detail code for creating MTStest.

Note

The following 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

```
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 callled 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
```

Example Code: ClientPrj for Distributed Transaction

The following code sample provides detail code for creating ClientPrj.

Note

The following 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 your 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
```

OCM Support for the COM Connector

You use J.D. Edwards Object Configuration Manager (OCM) to map business functions to an enterprise server so that COM 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. OCM mapping enables the COM interoperability server to distribute the processes of the COM connector client to various enterprise servers' requests, depending on the user, environment, and role name.

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

- Get the GenCOM ERP 9.0 (or later) version and regenerate the business wrapper function
- Configure your OCM and map your business function on the enterprise server.
- Add the following settings in your 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.

Your database administrator or ERP administrator can provide the following 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 your ODBC setting.
OCM Datasource = COM OCM	System data source for ERP 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 AS400, the separator is a "/".

If you use a client machine, the settings can be found in your client jde.ini file. The following examples shows the database name and object owner:

JDE9.SYS9

where JDE9 is the database name and SYS9 is the object owner

See Also

The following topics in the *Configurable Network Computing Implementation Guide*:

- *Object Configuration Manager* for more information about OCM
- *Mapping Objects* for information about mapping the OCM

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 only be installed on machines with the identical packages available.

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

Using COM Tracing and Logging

You can use COM tracing and logging to help you debug your COM applications. Tracing and logging are configurable through the jdeinterop.ini file. The logging format is similar to the ERP logging format. For example, both logging formats include the Time Thread ID [User ID] and Description.

> 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 DebugFile. 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 the following the steps:

- Modify the jdeinterop.ini file.
- Right-click the Connector System Tray Icon.
- Choose the menu item ChangeIniSettings.

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

Troubleshooting

Tracing affects performance. You do not need tracing turned on unless you are debugging. If your performance is negatively affected, ensure that you do not have your tracing level set to a nonzero number.

If no logs are generated, complete the following steps:

- Ensure that you have specified the proper path in the ini file.
- Verify that disk space and the permissions on your 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.

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, your 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 the user.

COM Connector Events

The J.D. Edwards COM connector events solution uses Microsoft's COM+ Events Service. COM+ Events Loosely Coupled Events, which matches and connects publishers and subscribers, is part of the Windows2000 Component Services. The EventClass is a COM+ component that contains interfaces and methods that are used by the publisher to fire events. The EventClass manages the connection between publisher and subscribers. J.D. Edwards provides the EventClass.dll, which contains the IOWEvent interface. The COM servers and COM clients must implement this interface so that when an event gets fired, this interface is called by the COM+ events 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.

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 is not dependent on the ERP set up for event generation.

See Also

- *Microsoft COM+* web site (*http://www.msdn.microsoft.com/*)
- *Events* in the *Interoperability Guide* for more information about generating Z, real-time, and XAPI events

Registering Components

So that subscribers can find an event class and subscribe to it, the ERP 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 ERP Events Class with COM+ Services, you must to the following:

- Add a new COM+ application for the ERP event class
- Install the ERP event class

Note

Before you register the ERP Event Class with COM+ Services, set up your COM server. Your COM server can be set up on either an ERP machine or a non-ERP machine, or both.

See Also

- *Installation Information* in the *Connectors Guide* for information about setting up your COM connector

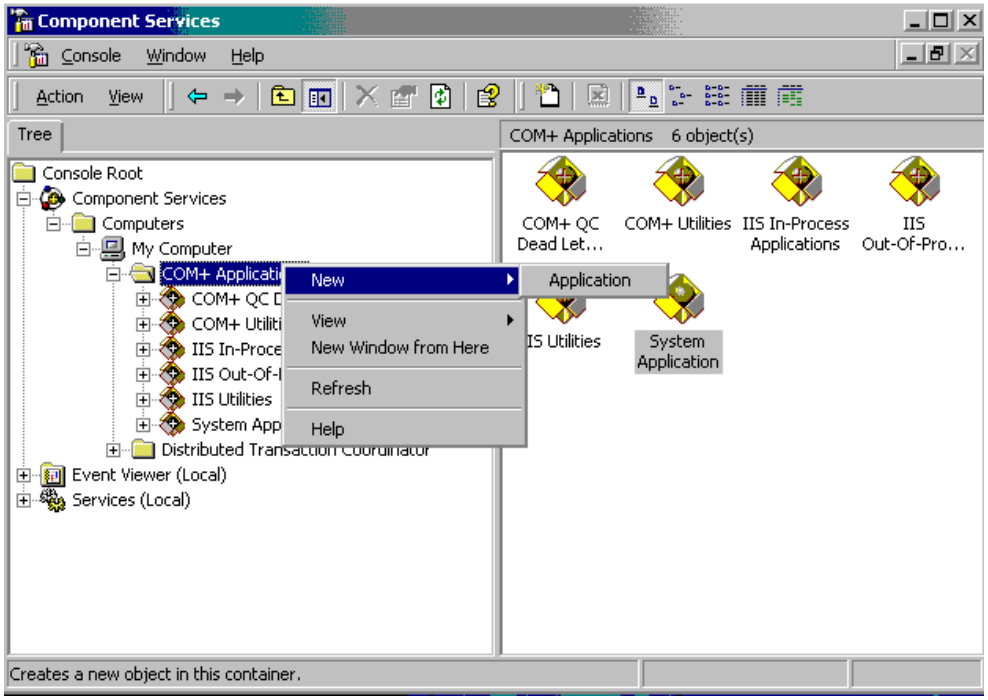
► To add a new application

From your Windows 2000 machine, navigate to COM+ Applications:

Control Panel > Administrative Tools > Component Services

Expand the following icons and folders:

Component Services > Computers > My Computer > COM+ Applications



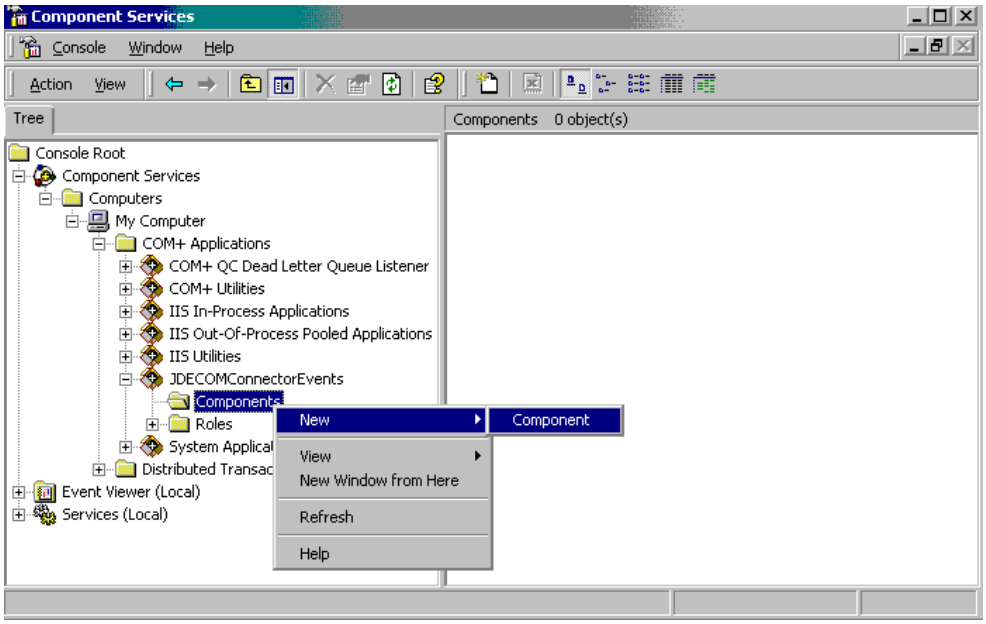
1. Choose COM+ Applications.
2. Right-click COM+ Applications, choose New, and then choose Application.

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

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

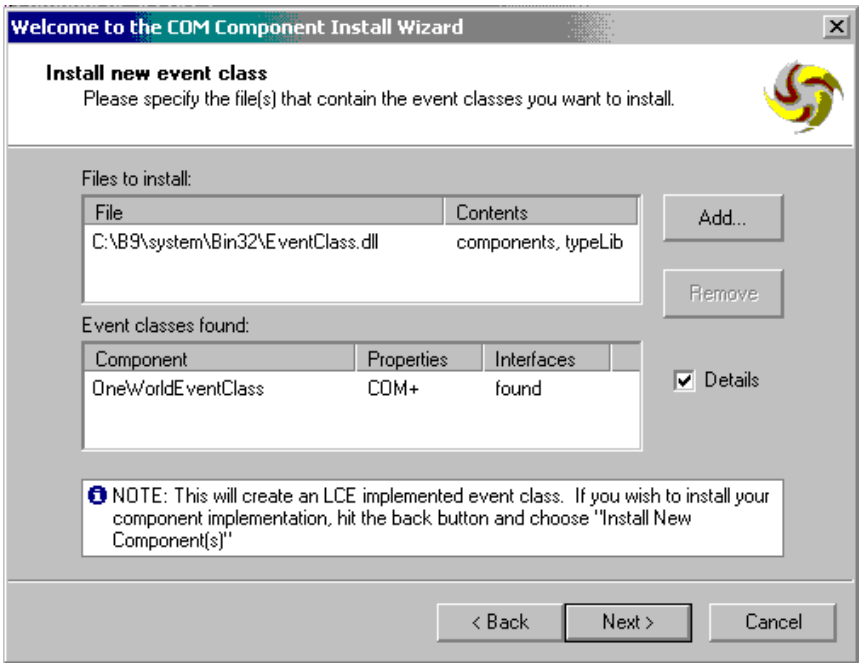
► To install the event class

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



1. Choose Components.
2. Right-click Components, choose New, and then choose Component.

   The COM Component Install Wizard appears. The following steps apply to the wizard.
3. On Import or Install a Component, choose Install new event class(es).
4. On Select Files to Install, browse to the EventClass.dll on your Windows 2000 machine.



5. Choose EventClass.dll, and then click Open.

   Install new event class appears with information in the following fields:

   - Files to install
   - Event classes found

6. Click Next, and then click Finish.

EventClass.dll is successfully added to Component Services.

Event Subscriptions

The COM connector supports both persistent and transient event subscriptions from the ERP enterprise server. The events are subscribed from the enterprise server that is specified in the [INTEROP] section of the jdeinterop.ini file. The events are received through the EventListener. The EventListener runs as long as the COM connector is up and running. The COM connector runs as a small globe icon in the bottom right corner of the Windows taskbar.

You must also set up the [EVENTS] section of the jdeinterop.ini file.

Note

The J.D. Edwards COM connector does not support subscription of events from multiple enterprise servers.

See Also

- *jdeinterop.ini* in the *Connectors Guide* for information about setting up the [INTEROP] and [EVENTS] sections

Logging COM Events

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

Implementing the ERP Interface

You must develop an object that implements the IOWEvent interface. For further discussion and to provide code samples, 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 ERP event. The EventSink implementation is client specific. EventSink receives the event from ERP by implementing the interface specified in EventClass. The following 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
```

The following 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 timeout 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.

Example: Sample Code for Creating EventSink

The following sample code is for creating the EventSink.dll. EventSink implements the EventClass interface, IOWEvent(). You can use a name other than EventSink.

EventSink: OneWorldTransientEventSink.cls

```
Option Explicit

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

Public Sub IOWEvent_OneWorldEvent(ByVal EventName As String, ByVal Data As
String)
    Dim flsObject As New Scripting.FileSystemObject
    Dim varEventFile As TextStream
    Dim strEventFile As String
    strEventFile = "C:\temp\eventDataPer.xml" ' change this to a valid
directory
    If Dir(strEventFile) = "" Then
        Set varEventFile = flsObject.CreateTextFile(strEventFile, False,
False)
    Else
        Set varEventFile = flsObject.OpenTextFile(strEventFile, ForWriting,
False)
    End If

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

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, do the following:

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

Note

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

► To install the EventSink component

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

1. Choose Components.
2. Right-click Components, choose New, and then choose Component.

   The COM Component Install Wizard appears. The following steps are for the wizard.

3. On Import or Install a Component, choose Install new component(s).
4. On Select Files to Install, browse to the EventSink.dll that you previously developed.
5. Choose EventSink.dll, and then click Open.

   Install new component appears with information in the following 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 the following folders:

JDECOMConnectorEvents > Components > EventSink.OneWorldTransientEventSink

1. Choose Subscription.
2. Right-click Subscription, choose New, and then choose Subscription.

   The COM New Subscription Wizard appears. The following steps apply to the wizard.
3. On Select Subscription Method(s), chose IOWEvent, and then click Next.
4. If appropriate, choose the following option:

   - Use all interfaces for this component
5. On Select Event Class, choose the event class (for example, PeopleSoft.EventClass.OneWorldEventClass.1), and then press Next.

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

   - Enable this subscription immediately
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 choose Properties.
10. On MySubscription Properties, click the Options tab.
11. Chose the following option:

    - Enabled
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'

Note
For more information about filtering, see *Subscribe parameter filtering* in the Microsoft COM+ Events online documentation.

13. Click OK.

Example: Sample Code for COMConnector.dll

The following 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

```
Option Explicit

Public bLoginEnv As Boolean

Private Sub cmdCancel_Click()
    'set the global var to false
    'to denote a failed login
    bLoginEnv = False
    Me.Hide
End Sub

Private Sub cmdOK_Click()
    'check for correct password
    If txtUserName = "" Or txtenvironment = "" Then
        bLoginEnv = False
        MsgBox "Must Enter User Name and Environment to continue"
    Else
        bLoginEnv = True
        Me.Hide
    End If
End Sub
```

COMConnector Common.bas

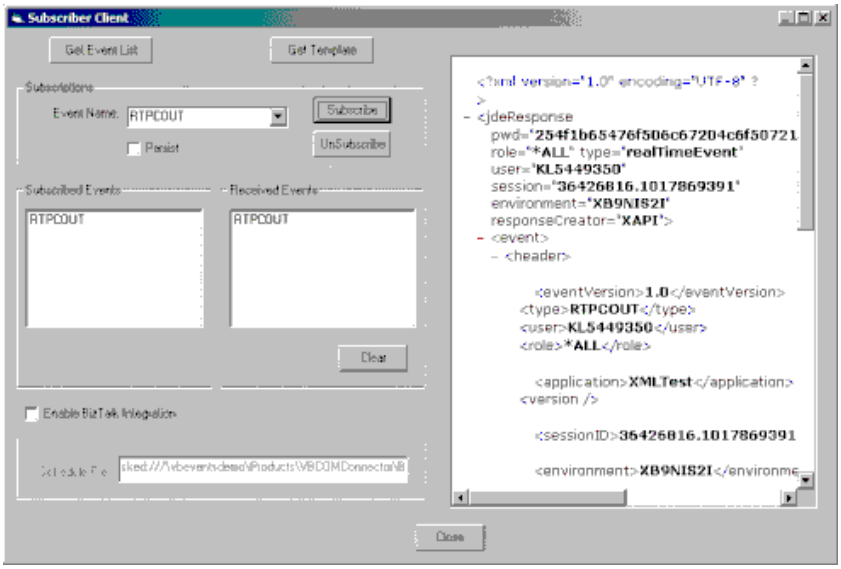
```
Option Explicit
Dim conn As New Connector
Dim connRole As IConnector2
Dim messageHandler As New messageHandler
Dim mHandlerInterface As ImessageHandler
Dim lngAccessNumber As Long
Public Sub comm_Initialize()
    Set connRole = conn
    frmLogin.bLoginEnv = False
    frmLogin.Show
    While Not frmLogin.bLoginEnv
        DoEvents
    Wend
    lngAccessNumber = connRole.Login(frmLogin.txtUserName,
frmLogin.txtPassword, frmLogin.txtenvironment, frmLogin.txtrole)
    Set mHandlerInterface = messageHandler
End Sub

' NOTE: the code is this module is particular to this prototype.  Different
code
' would be used in a production version to send messages to ERP via
' JDEdwards communication prototocols

Public Sub SendSubscriptionToOneWorld(eventName As String, oneworldevent As
IOWEvent, mode As Long)
    mHandlerInterface.SubscribeEvent lngAccessNumber, conn, eventName,
oneworldevent, mode
End Sub
Public Sub SendUnSubscribeToOneWorld(eventName As String, oneworldevent As
IOWEvent, mode As Long)
    mHandlerInterface.UnSubscribeEvent lngAccessNumber, conn, eventName,
oneworldevent, mode
End Sub
Public Sub getEventListFromOneWorld(eventList As String)
    mHandlerInterface.GetEventList lngAccessNumber, conn, eventList
End Sub
Public Sub getEventTemplateFromOneWorld(eventName As String, eventTemplate
As String)
    mHandlerInterface.GetEventTemplate lngAccessNumber, eventName, conn,
eventTemplate
End Sub
```

```
Option Explicit

Private m_SubscribedEvents As Collection

Private Sub Class_Initialize()
    Set m_SubscribedEvents = New Collection
    comm_Initialize
End Sub
Public Sub GetEventList(eventList As String)
    getEventListFromOneWorld eventList
End Sub

Public Sub CreateTransientSubscription(eventName As String, oneworldevent As
IOWEvent)
    SubscribeToOneWorldEvent eventName, oneworldevent, 0
End Sub
Public Sub CreatePersistentSubscription(eventName As String, oneworldevent
As IOWEvent)
    SubscribeToOneWorldEvent eventName, oneworldevent, 1
End Sub
Public Sub RemoveTransientSubscription(eventName As String, oneworldevent As
IOWEvent)
    UnSubscribeToOneWorldEvent eventName, oneworldevent, 0
End Sub
Public Sub RemovePersistentSubscription(eventName As String, oneworldevent
As IOWEvent)
    UnSubscribeToOneWorldEvent eventName, oneworldevent, 1
End Sub
Public Sub GetEventTemplate(eventName As String, eventTemplate As String)
    getEventTemplateFromOneWorld eventName, eventTemplate
End Sub
Public Sub SubscribeToOneWorldEvent(eventName As String, oneworldevent As
IOWEvent, mode As Long)
'Private Function SubscribeToOneWorldEvent(EventName As String) As Boolean
    ' we've already subscribed if the subscription is in our list
    Dim alreadySubscribed As Boolean
    alreadySubscribed = (CollectionContainsString(m_SubscribedEvents,
eventName) = True)

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

        ' ...and go ahead and subscribe to the event from ERP
        SendSubscriptionToOneWorld eventName, oneworldevent, mode
    End If

   'SubscribeToOneWorldEvent = alreadySubscribed
End Sub
```

```
Private Function CollectionContainsString(col As Collection, str As String)
    Dim colItem As Variant
    For Each colItem In col
        If (colItem = str) Then
            CollectionContainsString = True
            Exit Function
        End If
    Next
    CollectionContainsString = False
End Function


Public Sub UnSubscribeToOneWorldEvent(eventName As String, oneworldevent As
IOWEvent, mode As Long)
Dim alreadySubscribed As Boolean
     alreadySubscribed = (RemoveFromCollection(m_SubscribedEvents,
eventName))
        If (alreadySubscribed = False) Then
            MsgBox "Event Not Subscribed"
        Else
            ' ...and go ahead and subscribe to the event from ERP
            SendUnSubscribeToOneWorld eventName, oneworldevent, mode
        End If
  '  End If
End Sub
Private Function RemoveFromCollection(col As Collection, str As String)
Dim colItem As Variant
    Dim count As Integer
    count = 0
    For Each colItem In col
        count = count + 1
        If (colItem = str) Then
            col.Remove count
            RemoveFromCollection = True
            Exit Function
        End If
    Next
    RemoveFromCollection = False
End Function
```

Example: Sample Code for Subscriber

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 will populate the dropdown list box with the list of events that are supported by the enterprise server. Choose the event that needs to be subscribed from the enterprise server and the type of subscription. Click Subscribe to add a Subscription, or click Unsubscribe to unsubscribe from the enterprise server. As shown in the illustration, the Subscribed events and the Received events are shown 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

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

```
Option Explicit

' ---------------------------------- ** ------------------------------

'                              Member Variables

' ---------------------------------- ** ------------------------------

Private m_SubscriptionManager As SubscriptionManager
Private WithEvents m_OneWorldTransientEventSink As
OneWorldTransientEventSink
Private Sub Combo1_Change()

End Sub
Private Sub Check1_Click()

End Sub


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

'--------------------------------- ** ---------------------------------

'                           GetEventTemplate

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

    varTemplateFile.WriteLine EventTemplate
    varTemplateFile.Close

    wbEventData.Navigate "c:\temp\event_template.xml"
```

```
End Sub

' ---------------------------------- ** -----------------------------

'                                Event Handlers

' ---------------------------------- ** -----------------------------

Private Sub 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's 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
    myString = events
    Do Until events = ""
        If InStr(1, myString, ":") > 0 Then
            myValue = Left(myString, InStr(1, myString, ":") - 1)
            myString = Mid(myString, InStr(1, myString, ":") + 1)
        Else
            myValue = myString
            events = ""
        End If

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

'---------------------------------- ** ----------------------------------

'                           Subscribe Event

'---------------------------------- ** ----------------------------------

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

'---------------------------------- ** ----------------------------------

'                           UnSubscribe Event

'---------------------------------- ** ----------------------------------
Private Sub btnUnsubscribe_Click()
    Dim EventName As String
    EventName = cEventList.List(cEventList.ListIndex)
    Dim lstItem As ListItem
    Dim count As Integer
    Dim found As Boolean
    count = 0
    found = False
    For Each lstItem In lvwSubscribedEvents.ListItems
```

```
        count = count + 1
        If lstItem = EventName Then
            lvwSubscribedEvents.ListItems.remove (count)
            GoTo remove
            found = True
        End If
    Next
    If found = False Then
        MsgBox "Event Not Subscribed"
    End If
remove: If (chkPersist.Value = Checked) Then
        m_SubscriptionManager.RemovePersistentSubscription EventName,
m_OneWorldTransientEventSink
    Else
        m_SubscriptionManager.RemoveTransientSubscription EventName,
m_OneWorldTransientEventSink
    End If

End Sub


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()
    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
```

Example: Sample Code for BizTalk Integration

The following code is for the BizTalk integration for the received event.

Subscriber: BizTalk.cls

```
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 via 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 path to
your data are correct.  Also make sure the private queues have been
created." & vbCrLf & vbCrLf & Err.Description
    Exit Sub
End Sub
```

Java Interoperability Solution

The J.D. Edwards Java interoperability solution allows you to write Java applications that interact with the J.D. Edwards ERP system. The J.D. Edwards Java interoperability solution includes the following types of connectors:

- Dynamic Java Connector
- Java Connector
- Java Connector Architecture (JCA) Resource Adapter

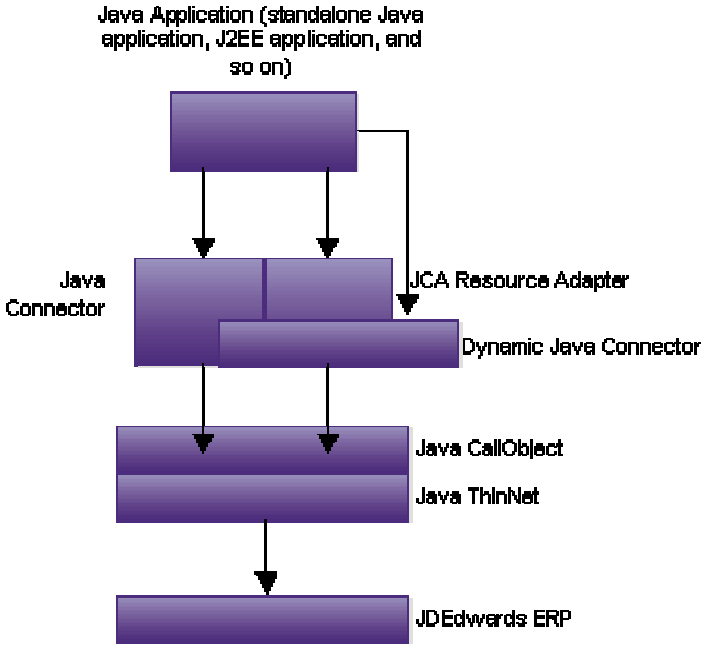
The initial Java interoperability solution provided by J.D. Edwards is the Java connector. The Java connector generates a Java wrapper object around the ERP business function and data structure. A Java application calls the ERP business functions from the Java wrapper object.

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

The JCA resource adapter is a thin layer built on top of the dynamic Java connector that provides standard APIs required by the Java connector architecture. The core functionality for the JCA resource adapter is to interact with ERP, and this functionality is leveraged to the dynamic Java connector.

Each connector has a complete set of APIs that allow Java applications to interact with ERP. You can choose the type of connector that best satisfies your purposes.

The following diagram shows how a Java application interacts with ERP through a connector:



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

Service	Description
Security Management	Handle security access to the ERP system.
User Session Management	Manage the user session pooling.
Business Function Calls	How the Java application calls ERP business functions.
Transaction Management	Manage the transaction process to the ERP system.
Error Handling	Provide the appropriate exceptions to the connector user to easily handle error scenarios.

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

Note

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

Dynamic Java Connector and ERP

The dynamic Java connector allows a Java application to call an ERP business function. Compared to the Java connector, the dynamic Java connector has the following distinguishing features:

- Dynamically introspects ERP business function metadata. The business function metadata is introspected from the ERP enterprise server during application design time by using connector APIs without pre-generating business function wrappers.
- Dynamically calls ERP business functions without pre-generating business function wrappers. Since there is no local storage of business function spec metadata, the business function used by the dynamic Java connector is always compatible with the server spec metadata.
- Easily switches from one environment to another environment. The Java application can run on any environment that is compatible to the environment on which the Java application was designed.

The dynamic Java connector provides the following services:

- For application design, the dynamic Java connector allows client programs to introspect ERP business function specification metadata.
- For application deployment, the dynamic Java connector validates whether a client application can run through a certain enterprise server.
- For application run time, the dynamic Java connector provides an interface that allows the connector client to call the ERP business function on the enterprise server.

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

Design Time for the Dynamic Java Connector

This section covers considerations for designing your dynamic Java connector.

Business Function Spec Metadata Introspection

To call an ERP 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. The following 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 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 allow you to retrieve metadata related to the business function method. The BSFNMethod interface defines the following APIs:

- public String getName();
- public String getDSTemplateName();
- public String getBSFNName();
- public String getDescription();
- public BSFNParameter getParamger(String paraName);
- public BSFNParameter[] getParameters();
- public String getFormatString();
- public ExecuatableMethod createExecuatable();
- 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 allow you to retrieve metadata related to the data structure of the business function. The BSFNParameter interface defines the following APIs:

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

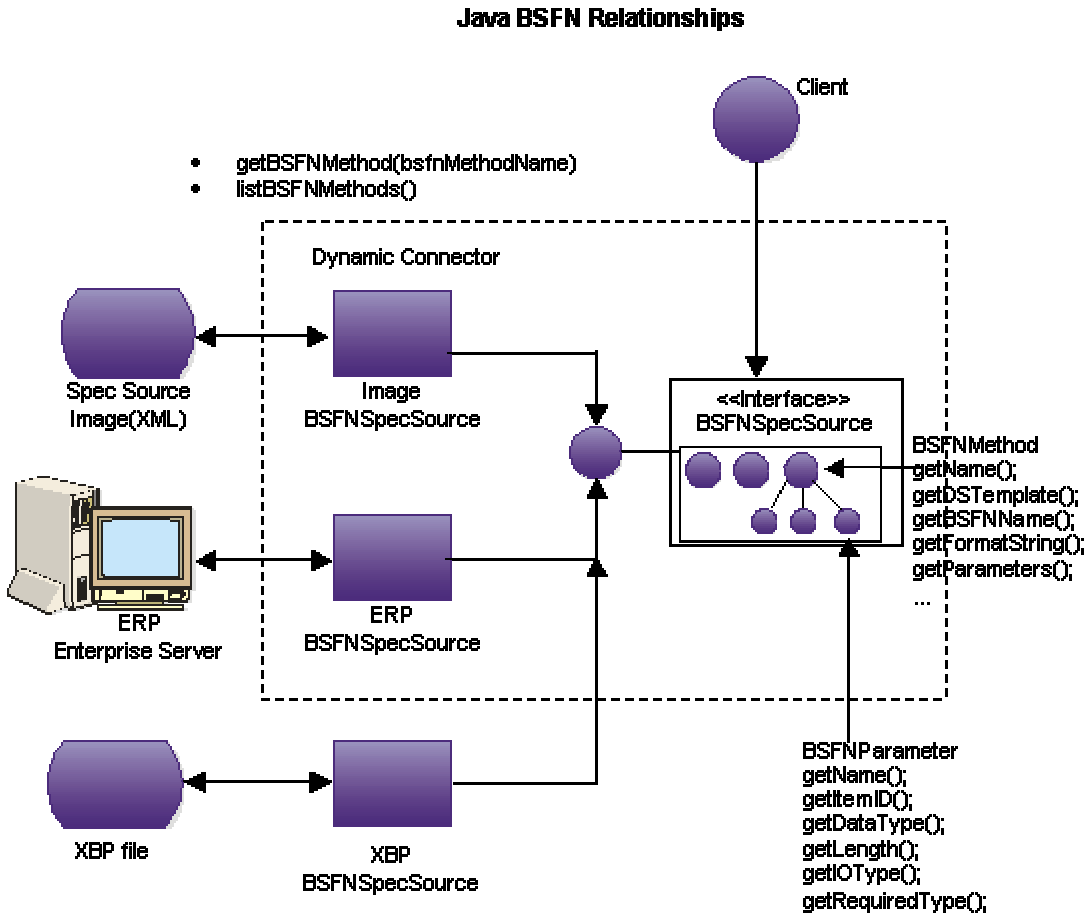
BSFNSpecSource

You can write a program to retrieve business function method metadata through an interface called BSFNSpecSource. The BSFNSpecSource interface defines the following 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.

The following illustration shows the BSFNSpecSource, BSFNMethod, and BSFNParameter relationships:



The following example of code shows how to retrieve the BSFN spec from BSFNSpecSource:

```
//Step 1: Create a new BSFNSpecSource
BSFNSpecSource specSource;
long sessionID = Connector.getInstance(). Login("user", "pwd", "env",
"role");
specSource = new OneWorldBSFNSpecSource(sessionID);
// or specSource = new ImageBSFNSpecSource("SSI.xml");

//Step 2: Get BSFNMethod by name from specSource
BSFNMethod method = specSource.getBSFNMethod("GetEffectiveAddress");

//Step 3: Introspect BSFNMethod metadata
method.getName();
…
BSFNParameter[] paraList = method.getParameters();
for (int I=0; I<paraList.length;i++) {
   BSFNParameter para = paraList[i];
   para.getName();
   para.getDataType();
…
}
```

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 the following services:

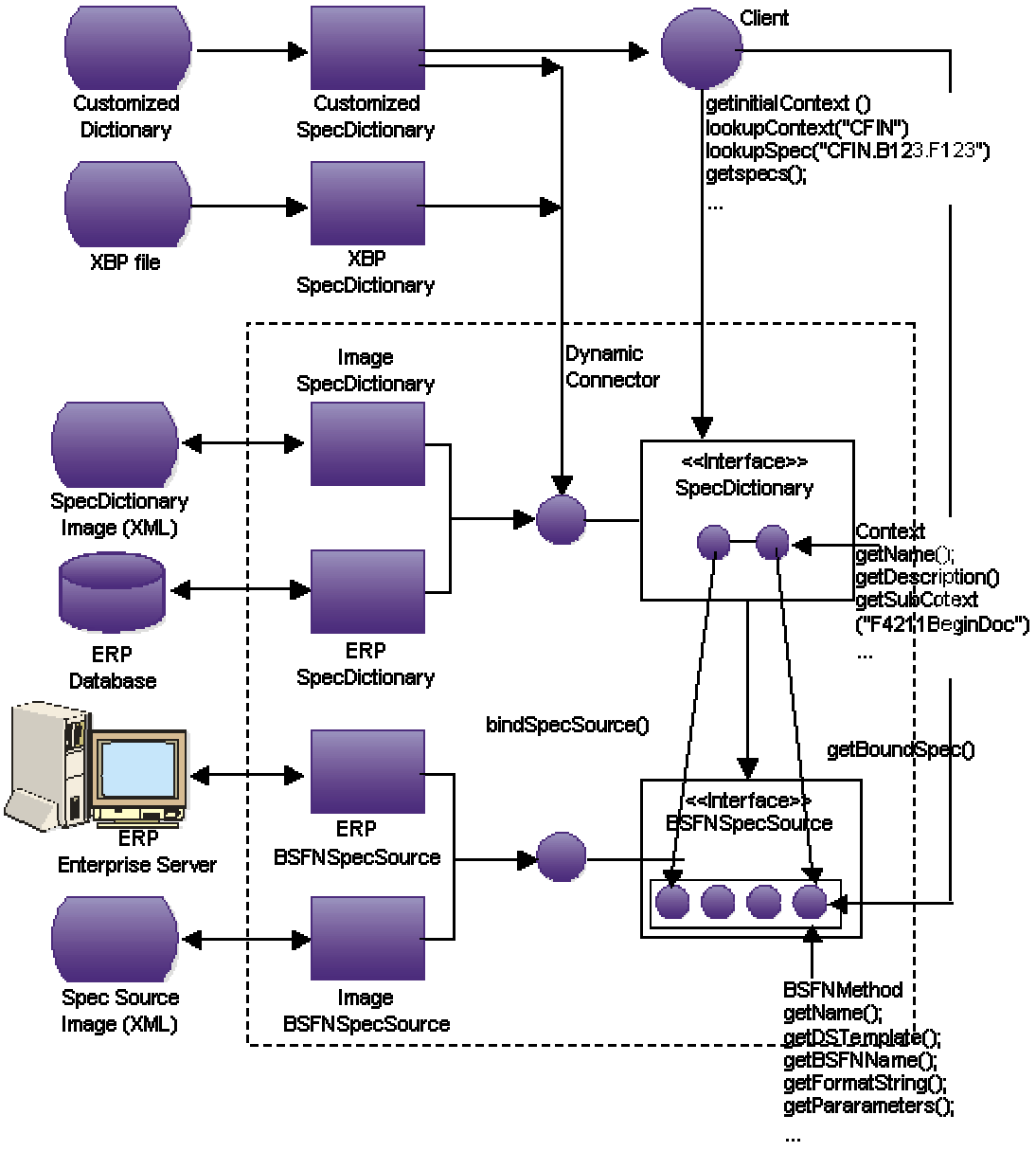
- Categorizes business function methods in a hierarchy.
- Masks the BSFNSpecSource and limits the number of business function methods a client can view

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 the following two concrete classes that implement the SpecDictionary:

- OneWorldSpecDictionary, which gets the hierarchy information from the ERP 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.

The following diagram shows the relationship between SpecDictionary and BSFNSpecSource:



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

```
BSFNSpecSource specSource;
SpecDictionary specDictionary;
//Step 1: Create a SpecDictionary
long 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
   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();
   for (int I=0;I<subContextList.length; I++) {
          Context subContext=subContext[i];
          subContext.getName();
          subContext.getDescription();
          method=(BSFNMethod)subContext.getBoundSpecContent();
          …
   }
```

Validate ERP Business Function Spec Metadata

If your 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 enterprise server where the business function runs. You must ensure that all input parameters are set correctly, according to OneWorldBSFNSpecSource.

If your 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 your local spec source might not be compatible with the business function metadata that is on the enterprise server where the business function will run. Local business function spec metadata can be validated during the following conditions:

Deploy Time	Your dynamic Java connector program validates your local spec source against the ERP enterprise server spec source before run time. J.D. Edwards recommends this validation, as all business functions in your local spec source are validated, and your program can be redesigned before it is shipped.
Run Time	The dynamic Java connector validates your 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 supplies two ways to validate business function spec metadata during deploy time: SpecImageValidator APIs and SpecImageConsole command line.

The APIs for SpecImageValidator are:

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

Note

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

Validating business function metadata from the SpecImageConsole command line is discussed later in this section.

Using the SpecImageConsole

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

Generate Spec Image

You use the spec image console to generate or regenerate a spec image. The following paragraphs provide information 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 <environmet> (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 ERP 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 will be written to the <output file> (or System.out if /OutputFile not present).

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

Example

```
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 can use the spec image console to update or change a spec image. The following paragraphs provide information 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 <environmet> (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 ERP 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

The following 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 can use the spec image console to validate the spec image against the ERP enterprise server. The following paragraphs provide information 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 <environmet> (required)

/Role <role> (required)

/SSI <SSI file> (required)

/SDI <SDI file> (optional)

/OutputFile (optional, default to System.out)

Explanation

Log on to ERP 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 will be written to the <output file> (or System.out if /OutputFile is not present).

Example

The following 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 can use the spec image console to synchronize the spec image with the ERP enterprise server. The following paragraphs provide information 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 <environmet> (required)

/Role <role> (required)

/SSI <SSI file> (required)

/SDI <SDI file> (optional)

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

Explanation

Log on to ERP 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 will be written to the <SSI file>. Error messages will be written to <err file> (or System.err if /ErrorFile is not present).

Example

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

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

Installing Dynamic Java Connector on a Non-ERP Environment

The following steps show how to install dynamic connector components so that you can run a dynamic Java connector application on a non-ERP machine.

1. Copy the following files from the enterprise server to a directory on the non-ERP machine that you want to use:
   - kernel.jar
   - connector.jar
   - database.jar
   - log4j.jar
   - xerces.jar
   - xalan.jar
   - jdeinterop.ini
   - jdeLog.properties
   - JDBC driver (you need to get JDBC driver from the vendor)

     For example, you might copy the above files to the following directory on your non-ERP machine:

     C:\PeopleSoft\Interop
2. Add the following files to the CLASSPATH:
   - kernel.jar
   - connector.jar
   - database.jar
   - log4j.jar
   - xerces.jar
   - xalan.jar JDBC driver
3. Add the path where the jdeLog.properties file is located into CLASSPATH.
4. Edit jdeinterop.ini and jdeLog.properties for proper settings.
5. Run the dynamic connector application with the system property config_file pointing to the location of the jdeinterop.ini file, for example:

   java -Dconfig_file=c:\PeopleSoft\Interop\jdeinterop.ini AddressBookApplication

See Also

- *jdeinterop.ini* in the *Connectors Guide* for more information about the jdeinterop.ini sections and settings

Run Time for the Dynamic Java Connector

This section covers run-time considerations for your dynamic Java connector.

Call an ERP Business Function

If you know the ERP 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. Unlike previous versions of the Java connector, the dynamic Java connector does not require pre-generated wrappers. The following code sample shows you how to call an ERP business function using the dynamic Java connector.

```
// Step 1: Login ERP
long sessionID = Connector.getInstance().login("user", "pwd", "env",
"role");

// Pre-condition: create the SpecDictionary or BSFNSpecSource as mentioned
before

// Step 2: Lookup the bsfn method from SpecDictionary or BSFNSpecSource as
metioned before
BSFNMethod bsfnMethod = (BSFNMethod)specContentSource.GetBSFNMethod ("Get
EffectiveAddress");

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

try {
// Step 4: Set parameter values
       addressbook.setValue("mnAddressbook", "1.0");
// Step 5: Execute the business function
       BSFNExecutionWarning warning = addressbook.execute(sessionID);
// Step 6: Get return parameter values
    System.out.println("szNamealpha=" + addressbook.getValueString
("szNamealpha"));
// Get the warnings if any
   if (warning.hasWarnings()){
       String warningMsgs[] = warning.getWarningMessages();
    for (int i=0;i<warningMsgs.length;i++){
              System.out.println(warningMsgs[i]);
   }
   }
}catch (SystemException e) {
       //SystemException is thrown when system crash, this is fatal
   // error and must be caught
   System.exit(1);
} catch (ApplicationException e){
   // ApplicationException is thrown when ERP business function
   // execution fail, this is RuntimeException and thus can be
   // unchecked. But it is strongly recommend to catch this
   // exception
```

```
}
//Log off and shut down connector if necessary
connector.logoff(sessionID);
connector.shutDown();
```

The dynamic Java connector allows you to use hash tables to input parameter values. The following example code illustrates inputting parameter values using the Hashtable class.

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

The dynamic Java connector allows you to use hash tables to retrieve output values. The following example code illustrates retrieving output values using the Hashtable class.

```
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 (ERP enterprise 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 ERP server the first time that 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 fetching 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 via Dynamic Java Connector

You can use the dynamic Java connector to do an ERP transaction in either auto or manual mode. The following example code for a purchase order entry transaction shows the steps for using the dynamic Java connector in manual mode.

```
long sessionID = Connector.getInstance().login("user", "pwd", "env",
"role");
UserSession userSession = Connector.getInstance().getUserSession (ses
sionID);
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 as
//mentioned before

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

//Step 4: run BSFNs in this transaction
//set poeBeginDoc input parameters as mentioned above, code not shown
BSFNExecutionWarning warning = poeBeginDoc.execute(transaction);
//set poeEditLine input parameters as mentioned above, code not shown
BSFNExecutionWarning warning = poeEditLine.execute(transaction);
//set poeEndDocinput parameters as mentioned above, code not shown
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 J.D. Edwards 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, you must do the following:

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

Ensure that the following settings in your jdeinterop.ini configuration file are set:

jdeinterop.ini File Section	Required Settings
OCM	OCMEnabled
JDBj-BOOTSTRAP SESSION	user, password, environment, and role
JDBj-BOOTSTRAP DATA SOURCE	name, databaseType, server, database, serverPort, physicalDatabase, library, owner
[JDBj-JDBC DRIVERS]	ORACLE, AS400, SQLSERVER, UDB
[JDBj-ORACLE]	tns

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 ERP business function calls, and the other is for outbound ERP 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 settings the connector looks at include the following:

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

The values for the settings are in milliseconds. A value of zero (0) indicates infinite time out. The above 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 ERP enterprise server, J.D. Edwards *highly* recommends that you explicitly call a connector API to log off the user session. The API log off releases the handle in the ERP enterprise server when the user session is no longer used.

The following sample codes shows how to retrieve and manage a user session:

```
// Login ERP
long sessionID = Connector.getInstance().login("user", "pwd", "env",
"role");

// Do whatever using the sessionID. If InvalidSessionException is caught,
user session is not valid any more

//Check the status of the usersession
UserSession session;
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.logoff(sessionID);
connector.shutDown();
```

Inbound XML Request via Dynamic Java Connector

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

The following example code shows how to use the Dynamic Java connector to execute an inbound XML request:

```
String xmlDoc;
//or byte[] xmlDoc
//…Load a String or byte[] into xmlDoc;

String requestResult;
try {
      XMLRequest xmlRequest = new XMLRequest(hostname, port, xmlDoc);
      requestResult = xmlRequest.execute();
} catch (IOException e) {
   System.out.println("Error in XML request");
}
//…handle requestResult.
```

See Also

See the following topics in the *Interoperability Guide*:

- *XML CallObject* for more information about using XML CallObject
- *XML List* for more information about using XML List
- *XML Transaction* for more information about using XML Transaction

Logging for the Dynamic Java Connector

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

- DEBUG
- INFO
- WARNING
- ERROR
- FATAL

The dynamic Java connector provides the following 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 show 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 according to your needs. For example, you might set log level to ERROR or FATAL for a production environment or to DEBUG for a development or test environment.

See Also

- *jdeLog.properties* in the *Connectors Guide* for more information about setting up logging configurations

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 the following 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 ERP tables), and SpecFailureException (unable to connect to Spec Source). It is up to the client program to catch this type of exception.

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

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

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

```
Public class  A
{
  public int foo1(DS 1 param, OneWorldInterface ow,
                  Connector c, int handle)
 {
  0
  }
  public int foo2(DS2 param, OneWorldInterface ow,
                  Connector c, int handle)
 {
  0
  }
  public DS1 Createfoo1ParameterSet()
 {
  0
  }
  public DS2 Createfoo2ParameterSet()
 {
  0
  }
  0
}
```

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

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

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

The data structure can contain two kinds of compound objects, JDEDate and JDEMathnumeric, in addition to the primitive data types. The two Java classes JDEDate and JDEMathnumeric are defined respectively. All public APIs can be found in the Java document ConnectorDoc.jar that is shipped with the product.

Note

If this is your first implementation of a Java connector, J.D. Edwards suggests you consider the Dynamic Java connector instead of the Java connector. The functionality is the same. The advantage of implementing the Dynamic Java connector is that you are not required to generate wrappers.

JDEDate

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

JDEMathNumeric

getValue()	Return the value as a String (for example, "-12345.6789")
setValue(String strValue)	Set the value from a String (for example, "-12345.6789")
getCurrencyDecimals()	Get the Currency Decimal positions
setCurrencyDecimals(int aValue)	Set the Currency Decimal positions
getCurrencyCode()	Get the Currency Code
setCurrencyCode(String aValue)	Set the Currency Code
getDecimalPosition()	Get the Decimal Position
isNegative()	Test if the value is negative
reset()	Reset all the internal values

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

Design Time for the Java Connector

This section covers considerations for designing your Java connector solution.

GenJava

J.D. Edwards provides a Java generation tool, GenJava, that you can run to expose business functions through Java. A system administrator usually runs GenJava.

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

Understanding Java Versioning

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

Migrating from Previous Releases

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

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

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

Java Connector Static and Dynamic Modes

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

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

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

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

Using the Java Wrapper Version Checker (CheckVer)

You can run CheckVer to verify whether a previously generated Java business object library is compatible with another environment. Typically, the system administrator performs this task. The XML files generated by GenJava are the signatures of the objects generated against specific ERP environments. These XML files can be used with CheckVer to verify that the wrappers in a previously generated jar file are compatible with these environments.

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

Running CheckVer (GenJava)

CheckVer is a Java class and should not be confused with the CheckVer.exe that is a part of the COM interoperability solution. CheckVer takes two arguments, the jar file name and the XML file name. CheckVer requires that the Connector.jar, Kernel.jar, xalan.jar, and xerces.jar files be in the CLASSPATH. This can be done either with the CLASSPATH environment variable or on the command line itself.

Syntax

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

Example

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

Setting Up a Client Environment for GenJava

You must take several steps to set up a client environment for GenJava. You should make sure the PATH environment variable and the CLASSPATH environment variable are set up correctly.

PATH

<bin directory for JDK>

Example: c:\jdk1.2.2\bin

CLASSPATH

<Directory where PeopleSoft ERP is located>\System\classes\Kernel.jar

<Directory where PeopleSoft ERP is located>\System\classes\Connector.jar

<Directory where PeopleSoft ERP is located>\System\classes\Xalan.jar

<Directory where PeopleSoft ERP is located>\System\classes\Xerces.jar

Installing Java Connector on a Non-ERP Environment

The following steps show how to install Java connector components so that you can run a Java connector application on a non-ERP machine

1. Copy the following files from the enterprise server to a directory on the desired machine. For example, copy the following files to c:\PeopleSoft\Interop on a non-ERP machine.
   - kernel.jar
   - connector.jar
   - jdeinterop.ini
   - xalan.jar
   - xerces.jar
   - database.jar
   - Log4j.jar
   - jdeLog.properties
   - JDBC driver (you need to get JDBC driver from the vendor)

     For example, you might copy the above files to the following directory on your non-ERP machine:

     C:\PeopleSoft\Interop
2. Add the following files to the CLASSPATH:
   - kernel.jar
   - connector.jar
   - database.jar
   - log4j.jar
   - xerces.jar
   - xalan.jar JDBC driver
3. Add the path where the jdeLog.properties file is located into CLASSPATH.
4. Create a separate repository directory for business object.jar files.
5. Run GenJava on the client machine and copy the output jar file (for example, JDEAddressBook.jar) to this directory.
6. Depending on whether you want the library in static mode or dynamic mode, put the business object.jar file in the CLASSPATH.
7. Run the Java application with the system property config_file pointing to the location of jdeinterop.ini. For example, java -Dconfig_file=c:\PeopleSoft\Interop\jdeinterop.ini AddressBookApplication.

Run Time for the Java Connector

This section covers run-time considerations for your Java connector.

Using the Java Generator (GenJava)

The J.D. Edwards Java generation tool, GenJava, provides access to business functions by generating Java interfaces for ERP business functions. GenJava includes the following components:

- GenJava.exe
- Emitter framework
- JDEIDAJavaEmitter.dll

You use the J.D. Edwards scripting language, iJDEScript, to script code generation activities when you use GenJava.

Running GenJava

You run GenJava from the command line. There are several options available for generation. GenJava is located in <install>\system\bin32.

Syntax

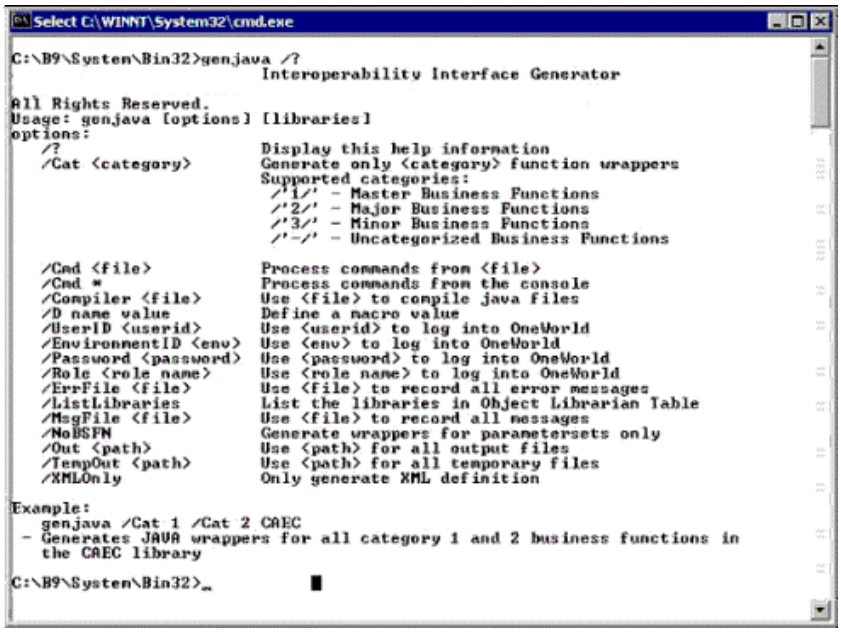
GenJava [options] [libraries]

Options

/?	Lists the options available for generation.
/Cat <category>	Generates only <category> function wrappers. Supported categories are: /'1/' - Master Business Functions /'2/' - Major Business Functions /'3/' - Minor Business Functions /'-/' - Uncategorized Business Functions
/Cmd *	Processes code generation commands from the console.
/Cmd <filename>	Processes code generation commands from <filename>.
/Compiler <file>	Uses <file> to compile Java files.
/D name value	Defines a macro value.
/EnvironmentID <env>	Uses <env> to log into ERP.
/ListLibraries	Lists the available libraries that you can use for GenJava.

/MsgFile <file>	Provides GenJava with the filename to log messages produced by GenJava during the generation process, for example, "messages.log".
/NoBSFN	Tells GenJava not to create wrappers for business functions. This option is for generating parameter sets only.
/Out <path>	Provides GenJava with the directory (path) in which to place the output files, for example "C:\winnt\system32".
/Password <password>	Provides GenJava with the password with which you want to log into ERP.
/Role	Provides GenJava with the role with which you want to log into ERP.
/TempOut <path>	Provides GenJava with the directory (path) in which to place temporary files needed for the build process, for example, "C:\temp".
/UserID <userid>	Provides GenJava with the username with which you wish to log into ERP.
/XMLOnly	Generate only the XML file.

The following illustration shows some of the available libraries you can use.



You can also use GenJava by running it with a JDEScript file, such as:

GenJava /cmd AddressBook.cmd

This prompts an ERP signon window for you to enter the user ID, password, role, and environment. The AddressBook.cmd is as follows:

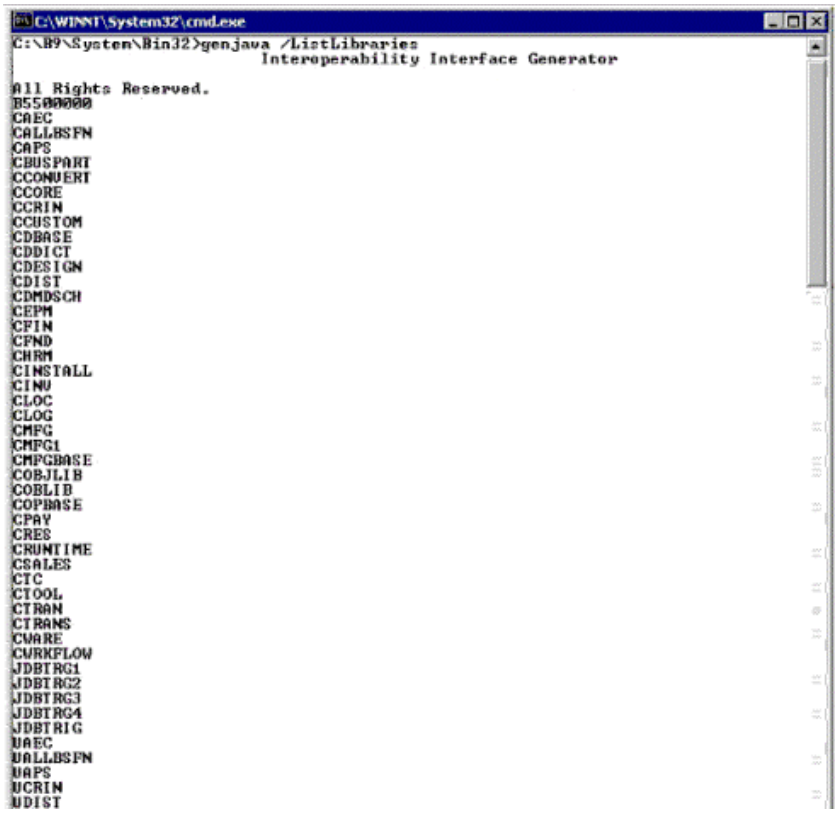
```
define library JDEAddressBook

login

library JDEAddressBook

interface AddressBook

import B0100031

import B0100019

import B0100032

import B0100002

import B0100033

build

logout
```

GenJava generates the wrappers in Java for all business functions imported in the script file.

Example: Generate Java Wrappers

GenJava /Cat 1 /UserID Devuser1 /Password Devuser1 /Environment ADEVHP02 CAEC

This example generates Java wrappers for Category 1 business functions in the CAEC library. You must use the correct information to log on to ERP, including the user ID, password, role, and environment.



Using GenJava Output

The output for GenJava produces fully functional Java objects based on the library you use to generate wrappers. GenJava packages these objects in a single jar file such as XXXXInterop.jar or XXXXInteropDoc.jar, where XXXX is the library name defined in the script file or from the command line. For example, JDEAddressBookInterop.jar is created for the above AddressBook.cmd. The default location for the jar file is under B9/System/classes at the drive of the client installed, but it can be somewhere else if you run GenJava using a /Out value. This jar file must be deployed to the machine that uses those wrappers. To import any wrapper object/class, the jar file must be added to the CLASSPATH. Because you are interacting with the ERP system, three components, Connector.jar, Kernel.jar and jdeinterop.ini file, must be deployed to the machine. A system property, config_file, must be set to point to jdeinterop.ini.

XXXXInteropDoc.jar is the compressed format of all the Java documents (html files) for all the classes generated by GenJava.unjar. You can also unzip, the jar file to see the APIs that can be called in these classes.

All Java client applications must do the following:

1. Initialize a com.jdedwards.system.connector.Connector.
2. Log in to ERP using a valid user ID, password, role, and environment name. The environment must be valid on the ERP enterprise server.

3. Get the OneWorldInterface object reference by calling Connector.CreateBusinessObject() with an object name, such as Connector::OneWorldInterface.
4. Get the object reference for the wrapper for the business function generated by GenJava, for example AddressBook. The object name passed into Connector.CreateBusinessObject should be "Library (Java package) Name:Object Name", such as "JDEAddressBook:AddressBook".
5. Call CreateXXXParameterSet() on the wrapper object for any data structure XXX.
6. Set the needed value in the data structure.
7. Call the business function with the data structure variable as a parameter. Check the return value. The return value can be one of the following:

   Successful = 0

   Warning = 1

   Error = 2

   Process the data returned by the business function.

8. Log off ERP.

The following examples illustrate how to use a generated Java business function wrapper in a Java application.

```
import com.jdedwards.system.connector.*;
import com.jdedwards.application.interop.jdeaddressbook.*;
public class abclient
{
   public static void main (String[] args) {
          Connector connectorProxy = null;
          OneWorldInterface ow;
          AddressBook ab;
          D0100033 ds;
          sessionID=0;
1.        connectorProxy = new Connector();
          try {
2.            sessionID = connectorProxy.Login("FOO", "BAR", "PDEVHPO2");
              System.out.printIn("Log in successfully");
          } catch (reject r) {
                System.out.printIn("got reject exception");
                String s = r.reason;
                System.out.printIn(s);
                System.exit(1);
          } catch (Exception e) {
                System.out.printIn("got other exception");
                e.printStackTrace();
                System.exit(1);
          }
          try {
3.            ow = (OneWorldInterface)connectorProxy.CreateBusiness Object
                        ("Connector:: OneWorldInterface", sessionID)
              System.out.printIn("got OneWorldInterface");
          } catch (reject r){
```

```
                String s = r.reason;
                System.out.printIn(s);
                return;
          }
          //create AddressBook object
          try {
4.            ab = (AddressBook)connectorProxy.CreateBusinessObject
                        ("JDEAddressBook:: AddressBook", sessionID)
              System.out.printIn("got AddressBook");
          } catch (reject r) {
                String s = r.reason;
                System.out.printIn(s);
                return;
          }
          // get data structure D0100033
5.        ds = ab.CreateGetEffectiveAddressParameterSet();
          // set addressbook number value in D0100033
6.        ds.getmnAddressNumber().setValue("1")
          // get address information
          int i = 0;
          try {
7.            i = ab.GetEffectiveAddress(ds, ow, connectorProxy; sessionID);
          } catch (reject e) { System.out.printIn(e.reason); }
          if (i!=2){
                String alphaname = ds.getszNamealpha();
                String address = ds.getszAddressLine1();
                String zipcode = ds.getszZipCodePostal();
                String city = ds.getszCity();
                String county = ds.getszCountyAddress();
                String state = ds.getszState();
                String country = ds.getszCountry();
                If (i==1){
                      System.out.printIn("warning count is"
                                +ow.GetWarningCount());
                      for ( int j = 0; j<ow.GetWarningCount(); j++) {
                            String s = ow.GetWarningAt(j);
                            System.out.printIn("warning" + j +";"+ s)
                      }
                }
          } else {
                for (int j = 0; j<ow.GetErrorCount(); j++){
                      String s = ow.GetErrorAt(j);
                      System.out.printIn("error" + j + ";" + s);
                }
                System.out.printIn("BSFN error");

                //log off
8.              connectorProxy.Logoff(1);
      } // end main
}  // end abclient
```

Transaction via Java Connector

Transactions are a way to update the ERP database. You can use the Java connector to do a transaction in either auto mode or manual mode. When you use auto transaction mode, the transaction is immediately committed after the business function call is completed. The transaction is set to the auto commit mode by default. When you use manual transaction mode, the transaction is started by explicitly calling BeginTransaction in OWInterface, and the transaction is committed (or rolled back) by calling Commit(or Rollback) in OWInterface.

Note

The ERP transaction is not really a two-phase commit. You need to manually roll back the transaction when the commit statement is reached.

The following example shows a basic manual commit transaction:

```
import com.jdedwards.system.connector.*;

public class ConnectorDemo {
    public static void main(String argv[]) {
    OWInterface ow;
    try {
    Connector con = new Connector();
           int accessnumber = con.login("User","Password","Env","Role");
           ow = (OneWorldInterface)
con.CreateBusinessObject("Connector::OneWorldInterface", 1);
           // ... handle the message
           ow.BeginTransaction(con, accessNumber);
           soe.F4211FSBeginDoc(soeBeginDoc,ow, con, accessnumber);
           soe.F4211FSEditLineDoc(soeEditLine,ow, con, accessnumber);
           soe.F4211FSEditLineDoc(soeEditLine,ow, con, accessnumber);
           soe.F4211FSEndDoc(soeEndDoc,ow, con, accessnumber);
           ow.Commit();
    }catch (Exception e) {
           ow.rollback();
    }
}
```

Using BHVRCOM via Java Connector

You can use the BHVRCOM structure to control the execution of business functions. You use the Java connector to call methods in the OWInterface class to set and pass the BHVRCOM fields to business functions on the server. The following table shows the business function methods and the BHVRCOM fields:

Business Function Method	BHVRCOM Field
setBOBMode(int bobMode)	lBobMode
setAPPName(StringaName)	szApplication
setUserName(String aName)	szUser

setDatabaseChanged(Boolean value)	bDataBaseChange

The following Java code demonstrates how to query the IBHVRCOM interface and pass values to business functions:

```
…
    ow = (OneWorldInterface)
connectorProxy.CreateBusinessObject("Connector::OneWorld Interface", l);
ab=(AddressBook)connectorProxy.CreateBusinessObject("JDEAddress
Book::Address Book", l);
ds.getmnAddressNumber().setValue("1");
ow.setAppName("AddressbookApp");
ow.setBOBMode(8);
ow.setUserName("Java Connector");
ow. SetDatabaseChanged(false);
    i = ab.GetEffectiveAddress(ds, ow, connectorProxy, l);
…
```

OCM Support for the Java Connector

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

- Use a B9 or later version of GenJava to regenerate the business wrapper function
- Configure your OCM and map your business function on different enterprise servers.
- Set OCMEnabled=true in jdeinterop.ini
- Configure the settings in jdeinterop.ini regarding the bootstrap data source with your OCM configuration.

Ensure that the following settings in your jdeinterop.ini configuration file are set:

jdeinterop.ini File Section	**Required Settings**
OCM	OCMEnabled
JDBj-BOOTSTRAP SESSION	user, password, environment, and role
JDBj-BOOTSTRAP DATA SOURCE	name, databaseType, server, database, serverPort, physicalDatabase, library, owner
[JDBj-JDBC DRIVERS]	ORACLE, AS400, SQLSERVER, UDB
[JDBj-ORACLE]	tns

The dynamic Java connector and the Java connector use OCM in the same way.

User Session Management for the Java Connector

When the connector user successfully signs on, a valid user session is allocated to that user sign on. The user session has status for two types of connector operations, one is for inbound ERP business function calls, and the other is for outbound ERP 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 settings the connector looks at include the following:

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

The value for the settings is in milliseconds. A value of zero (0) indicates infinite time out. The above 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 ERP enterprise server, J.D. Edwards *highly* recommends that you explicitly call a Connector API to log off the user session to release the handle in the ERP enterprise server when the user session is no longer used.

The following sample codes shows how to retrieve and manage a user session:

```
// Login ERP
long sessionID = Connector.getInstance().login("user", "pwd", "env",
"role");

// Do whatever using the sessionID. If InvalidSessionException is caught,
user session is not valid any more

//Check the status of the usersession
UserSession session;
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.logoff(sessionID);
connector.shutDown();
```

The dynamic Java connector and the Java connector handle XML requests in the same way.

Inbound XML Request via Java Connector

You can use the Java connector to send inbound synchronous XML requests (such as XML CallObject and XML List) to the ERP server. The Java connector has an API that it calls to send XML documents to JDENET.

The following example code shows how to use the Java connector to execute an inbound XML request:

```
Connector conn =  new Connector();
//…login into OW
String xmlDoc;
//or byte[] xmlDoc
//…Load a String or byte[] into xmlDoc;

String requestResult = conn.executeXMLRequest(xmlDoc);
//…handle requestResult.
```

The Dynamic Java connector and the Java connector handle XML requests in the same way.

See Also

See the following topics in the *Interoperability Guide*:

- *XML CallObject* for more information about using XML CallObject
- *XML List* for more information about using XML List
- *XML Transaction* for more information about using XML Transaction

Exception Handling for the Java Connector

When you run the Java connector or the GenJava tool, the program might encounter a condition that causes unexpected results or system failure. When the program does not perform as expected, an error occurs; or, using Java terminology, an exception is thrown. In Java, the system, classes, and programs can throw exceptions. You can write code to catch exceptions. Catching an exception involves dealing with the exception conditions so that your program will not crash.

All exceptions in the connector and GenJava code inherit from the reject class. Your program needs to catch only the reject exception conditions for the methods that throw exceptions. J.D. Edwards created the FatalException class and the RecoverableException class so that you can provide a recovery action in your program for some exceptions, thereby minimizing manual intervention.

Fatal Exception

FatalException class conditions are unlikely or impossible to resolve without manual intervention. If you catch fatal exception conditions in your program, you can include a string message that indicates the condition that occurred. You use the getMessage() method from the java.lang.Throwable class to retrieve fatal exception messages from your program. The system uses the INTEROP category to log fatal exception conditions to the jas.log file.

Recoverable Exception

You can provide the capability for the system to possibly resolve an exception condition by catching RecoverableException (and children) class conditions in your program. The children of recoverable exception conditions indicate through their class names the category of the exception and include a sting message in the constructor to provide more exception details. You use the getMessage() method from the java.lang.Throwable class to retrieve recoverable exception messages from your program. The system uses the INTEROP category to log recoverable exception conditions to the jasdebug.log file. You can turn off recoverable exception messages through the DEBUG flag in the jdeinterop.ini file. The flag is either true or false.

Reject

The method signature for each of the methods listed in the following table indicates that the method only throws “reject,” even though the exceptions thrown in each method’s code are children of the reject class. Even if you decide to catch all of the exceptions listed in Exception Details table which follows, you will also need to catch “reject” as the last in the series of connector-related catch statements because of the method signatures’ stated throws clause.

Exception Details

The methods that throw exceptions in each of the main public classes of the connector (Connector, OneWorldInterface, EventSource, and GenJava-created business object code) are detailed in the following table. The information in this table is also available in the Javadoc for the connector, which is in the ConnectorDoc.jar file.

Class	Method	Exception	Condition	Possible Action
Connector	Login	CallObjectRetryException	The error code returned by CallObject is TIMEOUT or RETRY_NEEDED	Retry Login method
		CallObjectIgnoreException	The error code returned by CallOjbect is NOERROR, ALREADY_EXECUTED, or BAD_ERRORPACKETS	Ignore this exception
		FatalException	The error code returned by CallObject is any other error code	*
	CreateBusinessObject	NotLoggedInException	The user is not currently logged in to ERP	Login through Connector class
		FatalException	A Java reflection exception is thrown or the ERP environment is not in sync with the business function wrapper	*
OneWorldInterface	GetNextError	NoMoreDataException	Error index reaches the end of the array	End the loop searching for the next error
	GetNextWarning	NoMoreDataException	Warning index reaches the end of the array	End the loop searching for the next warning
	Commit	InvalidMethodCallException	This method is called before PrepareToCommit() is called	Call the PrepareToCommit() method
		CallObjectRetryException	The error code returned by CallObject is TIMEOUT or RETRY_NEEDED	Retry Commit method
		CallObjectIgnoreException	The error code returned by CallObject is NOERROR, ALREADY_EXECUTED, or BAD_ERRORPACKETS	Ignore this exception
		FatalException	The error code returned by CallObject is any other error code	*
	Rollback	CallObjectRetryException	The error code returned by CallObject is TIMEOUT or RETRY_NEEDED	Retry Rollback method

Class	Method	Exception	Condition	Possible Action
		CallObjectIgnoreException	The error code returned by CallObject is NOERROR, ALREADY_EXECUTED, or BAD_ERRORPACKETS	Ignore this exception
		FatalException	The error code returned by CallObject is any other error code	*
	PrepareToCommit	CallObjectRetryException	The error code returned by CallObject is TIMEOUT or RETRY_NEEDED	Retry PrepareToCommit method
		CallObjectIgnore Exception	The error code returned by CallObject is NOERROR, ALREADY_EXECUTED, or BAD_ERRORPACKETS	Ignore this exception
		FatalException	The error code returned by CallObject is any other error code	*
	ExecuteBSFN	NotLoggedInException	The user is not currently logged in to ERP	Login through Connector class
		CallObjectRetryException	The error code returned by CallObject is TIMEOUT or RETRY_NEEDED	Retry ExecuteBSFN method
		CallObjectIgnoreException	The error code returned by CallObject is NOERROR, ALREADY_EXECUTED, or BAD_ERRORPACKETS	Ignore this exception
		FatalException	The error code returned by CallObject is any other error code	*
EventSource	EventSource (Constructor)	FatalException	The connector cannot listen on the given port	*
	addListener	NotLoggedInException	The user is not currently logged in to ERP	Login through Connector class
		FatalException	The subscription fails	*
	removeListener	NotLoggedInException	The user is not currently logged in to ERP	Login through Connector class
		FatalException	The unsubscription fails	*

Class	Method	Exception	Condition	Possible Action
	updateSession	NotLoggedInException	The user is not currently logged in to ERP	Login through Connector class
	getEventTemplate	NotLoggedInException	The user is not currently logged in to ERP	Login through Connector class
		FatalException	A JdeNetException is thrown	*
	getEventTypes	NotLoggedInException	The user is not currently logged on to ERP	Login through Connector class
		FatalException	A JdeNetException is thrown	*
GenJava-created Data Structures	setString<parameter> methods	StringTooLongException	The value set for the parameter is too long	Reset the parameter using a shorter length

For FatalException conditions, you can send the exception message, which can be retried by using the getMessage() method, to your system administrator. Alternatively, you can prompt your system administrator to look in the jas.log file for more details about the exception. It is unlikely that your program can recover associated ERP or connector errors during runtime.

Example: Java Connector Exception Handling Sample Code

The following code illustrates some of the features of the enhanced connector exception handling. The bold-faced items indicate specific exception-handling code.

```
import com.jdedwards.system.connector.*;
import com.jdedwards.application.interop.jdeaddressbook.*;

public class AddressClient {
  public static void main(String[] args) {
    if (args.length != 1) {
      System.out.println("Must supply a city to query for AddressBook");
      System.exit(-1);
    }

    Connector connectorProxy = null;
    OneWorldInterface ow = null;
    AddressBook ab = null;
    D0100033 ds = null;

    int accessNumber = 0;
    connectorProxy = new Connector();

    try {
      accessNumber = connectorProxy.Login("FOO", "BAR", "PDEVHP02");
      System.out.println("Logged in successfully");
    } catch (CallObjectIgnoreException e) {
      // do nothing
    } catch (CallObjectRetryException e) {
```

```
      // try one more time
      try {
        accessNumber = connectorProxy.Login("FOO", "BAR", "PDEVHP02");
        System.out.println("Logged in successfully");
      } catch (CallObjectIgnoreException ex) {
        // do nothing
      } catch (CallObjectRetryException ex) {
        System.out.println("EXCEPTION: " + ex.toString());
        System.out.println("Nested Exception: " +
ex.getChainedException().toString());
        System.out.println("Refer to the jasdebug.log file for more
details.");
        System.exit(-1);
      } catch (FatalException ex) {
        System.out.println("Fatal Exception during login: " +
ex.toString());
        System.out.println("Refer to the jas.log file for more details.");
        System.exit(-1);
      } catch (reject r) {
        System.out.println("Java Connector Exception: " + r.reason);
        System.exit(-1);
      }
    } catch (FatalException e) {
      System.out.println("Fatal Exception during login: " + e.toString());
      System.out.println("Refer to the jas.log file for more details.");
      System.exit(-1);
    } catch (reject r) {
      /* This should not happen, as the Java Connector code
       * now only throws one of the reject child objects.
       * The documentation indicates which methods throw which
       * reject child exception objects.  All methods continue
       * to have a signature of "throws reject", however, for
       * backwards compatibility (to not break existing client code).
       */
      System.out.println("Java Connector Exception: " + r.reason);
      System.exit(-1);
    }

    try {
      ow =
(OneWorldInterface)connectorProxy.CreateBusinessObject("Connector::OneWorldI
nterface", accessNumber);
      System.out.println("Got OneWorldInterface");
    } catch (FatalException e) {
       System.out.println("Fatal Exception during OneWorldInterface
creation: " + e.toString());
       System.out.println("Refer to the jas.log file for more details.");
      System.exit(-1);
    } catch (reject r) {
      System.out.println("Java Connector Exception: " + r.reason);
      System.exit(-1);
    }


    try {
```

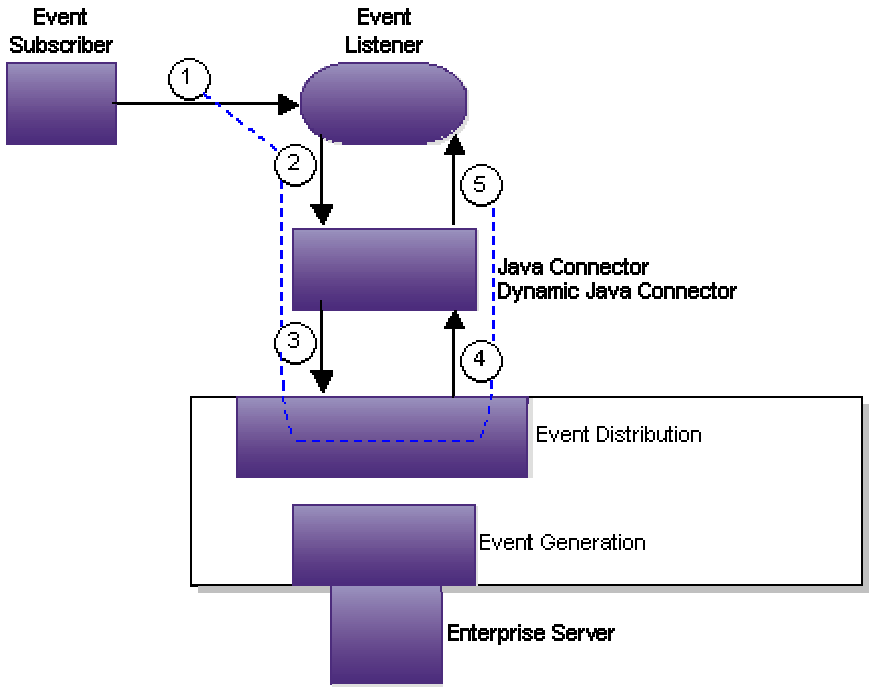
```
      ab =
(AddressBook)connectorProxy.CreateBusinessObject("JDEAddressBook::AddressBoo
k", accessNumber);
      System.out.println("Got AddressBook");
    } catch (FatalException e) {
       System.out.println("Fatal Exception during OneWorldInterface
creation: " + e.toString());
       System.out.println("Refer to the jas.log file for more details.");
      System.exit(-1);
    } catch (reject r) {
      System.out.println("Java Connector Exception: " + r.reason);
      System.exit(-1);
    }

    ds = ab.CreateGetEffectiveAddressParameterSet();

    ds.getmnAddressNumber().setValue("1");

    try {
      ds.setszCity(args[0]);
    } catch(StringTooLongException e) {
      System.out.println("Cannot set a city with length of " +
args[0].length());
      System.exit(-1);
    } catch (reject r) {
      System.out.println("Java Connector Exception: " + r.reason);
      System.exit(-1);
    }

    int i=0;

    try {
      i = ab.GetEffectiveAddress(ds, ow, connectorProxy, accessNumber);
    } catch (CallObjectIgnoreException e) {
      // do nothing
    } catch (CallObjectRetryException e) {
      // try one more time
      try {
        i = ab.GetEffectiveAddress(ds, ow, connectorProxy, accessNumber);
      } catch (CallObjectIgnoreException ex) {
        // do nothing
      } catch (CallObjectRetryException ex) {
        // don't try again after second try
        System.out.println("EXCEPTION: " + ex.toString());
        System.out.println("Nested Exception: " +
ex.getChainedException().toString());
        System.out.println("Refer to the jasdebug.log file for more
details.");
        System.exit(-1);
      } catch (FatalException ex) {
        System.out.println("Fatal Exception during AddressBook retrieval: "
+ ex.toString());
        System.out.println("Refer to the jas.log file for more details.");
        System.exit(-1);
      } catch (reject r) {
```

```
        System.out.println("Java Connector Exception: " + r.reason);
        System.exit(-1);
      }
    } catch (FatalException e) {
      System.out.println("Fatal Exception during AddressBook retrieval: " +
e.toString());
      System.out.println("Refer to the jas.log file for more details.");
      System.exit(-1);
    } catch (reject r) {
      System.out.println("Java Connector Exception: " + r.reason);
      System.exit(-1);
    }

    String alphaname = ds.getszNamealpha();
    String address   = ds.getszAddressLine1();
    // get whatever other AddressBook parameters you desire...

    if (i == 1) { // business function warning
      System.out.println("Warning count is " + ow.GetWarningCount());
      for (int j=0; j<ow.GetWarningCount(); j++) {
      } System.out.println("Warning " + j + ": " + ow.GetWarningAt(j));
    } else if (i == 2) { // business function error
      for (int j=0; j<ow.GetErrorCount(); j++) {
        System.out.println("Error " + j + ": " + ow.GetErrorAt(j));
      }
    }
    connectorProxy.Logoff(accessNumber);
  }
}
```

Java Connector Outbound Events

The Java connector outbound event source architecture enables Java clients to use either the Java connector or the dynamic Java connector to subscribe to various transaction types in ERP and receive notification upon completion of those transactions. For example, a client can subscribe to the event, JDESOOUT, and then receive notification when a sales order transaction is complete in ERP.

The following diagram illustrates this process:



1. ERP clients create different types of EventListeners.
2. ERP clients subscribe to various event types with the Java connector.
3. When the Java connector receives a subscription for a given event, it subscribes to the same event type with the event distribution kernel.
4. ERP events originate from the real-time events kernel or from callback functions in Uses. When the event distribution kernel receives an event to which the Java connector has subscribed, it sends the event to the Java connector.
5. The Java connector sends the event to all subscribers for that event. The EventListener callback function is executed to receive the subscribed event.

Note

The outbound events architecture is the same for the Java connector and the dynamic Java connector. The difference is that corresponding package location for the dynamic Java connector is comjdedwards.system.connector.dynamic.* and com.jdedwards.system.connector.dynamic.events.*.

For purposes of discussion in this document, the Java connector is used to illustrate the outbound events architecture. Special notes are added to discuss any API differences between the Java connector and dynamic Java connector.

Do not mix the usage of APIs from the two connectors in one application.

Developing the Java Client to Use the Java Connector Outbound Event Source

You can use the Java connector outbound event source to subscribe to an ERP outbound event. The following list identifies the tasks for setting up and using your Java client to subscribe to ERP transaction types and notify you upon completion of the transaction.

- Create a Java class to implement an interface
- Create a Java client application to subscribe to an ERP event
- Compile and run the Java client

Implement an Interface

You create a Java class to implement an interface to ERP. Depending on the purpose for which you are using the Java class, implement one of the following interfaces:

- com.jdedwards.system.connector.events.CountedListener

  If you want to know the subscription count, when the subscription count is reached, and when the subscribed event is dropped, implement this interface.

- com.jdedwards.system.connector.events.PersistentListener

  If you want the real-time event kernel to persist the subscription when the kernel goes down and comes up and the connection to the Java connector is re-established, implement this interface.

- com.jdedwards.system.connector.events.EventListener

  For or most other situations, implement this interface.

No matter which interface you implement, the implementation Java class must contain the following five methods:

```
//set the event type to subscribe
void setEventType( String type );
String getEventType();

// stop/start the event coming in the Java connector
void setPause( boolean pause );
boolean isPaused();

// the callback function when the event arrives
void onOneWorldEvent( EventObject event );
```

Subscribe to an ERP Event

You create a Java Client Application to subscribe to an ERP event. Your Java Client Application must do the following:

1. Create a new instance of the Connector class.
2. Use the Connector object to verify the client's user ID, password, and environment, and then log the client onto ERP.
3. Do one of the following:
   - Java Connector: Create an EventSource object by calling the CreateBusinessObject method of the Connector class, passing in an "Events::EventSource" string identifier.
   - Dynamic Java Connector: Get EventSource instance by calling com.jdedwards.system.connector.dynamic.connector.events.EventSource.getInstance().
4. Create an EventListener object.
5. Specify the specific event type to which to subscribe.
6. Register the EventListener object with the EventSrc object.
7. Develop a callback function. When the subscribed to event arrives, the EventListener calls this callback function. This step is optional.

Example: Using the Java Client to Subscribe to an ERP Event Using the Java Connector Outbound Event Source

The following example illustrates how to write code for a Java client to subscribe to an ERP event using the Java connector outbound event source.

```
import java.io.*;
import javax.swing.*;
import com.jdedwards.system.connector.*;
import com.jdedwards.system.connector.events.EventListener;

/**
* The event source client application
*/
class EventClient
{
```

```
   private Connector m_connector = null;
   private int m_Access = 0;
   private EventSource m_theSource = null;
   private Listener m_listener = null;

   public static void main(String argv[]) {
         try
          {

               // 1.
            m_connector = new Connector();

               // 2.
            m_Access = m_connector.Login("user", "password",
"environment");

               // 3.
               // passing in an "Events::EventSource" string
identifier.
            m_theSource = (EventSource)m_connector.CreateBusiness
Object("Events:: EventSource", m_Access);

               // 4.
                     m_listener = new ListenerImpl(this);

               // 5.
                     m_listener.setType("JDESOOUT");

               // 6.
            m_theSource.addListener( m_listener, m_Access );
         }
         catch(Exception e)
         {
            System.out.println(e.toString())
            System.out.println(e.getMessage());
            e.printStackTrace();
         }
   }

   // 7.
   public synchronized void executeCallBack(EventObject event){
         System.out.println("Getting the event:"+event.getData());
         //execute the call back function;
   }

}

/**
* The EventListener interface is the means by which events are
* delivered to the client by the Java connector.
* The client have to implement an EventListener object
*/

public class ListenerImpl implements EventListener
{
```

```
   String m_eventType;
   boolean m_paused = false;
   EventClient m_client;

   /** Creates new Listener */
   public ListenerImpl()
   {
   }
   public ListenerImpl(EventClient client)
   {            this.m_client=client;
   }

   public synchronized String getEventType()

   {
   return m_eventType;
   }

   public void setEventType(java.lang.String eventType)
   {
    this.m_eventType = eventType;
   }

   public synchronized boolean isPaused()
   {
      return m_paused;
   }
   public synchronized void setPause(boolean pause)
   {
      m_paused = pause;
   }

   public synchronized void onOneWorldEvent(EventObject p1)
   {
      System.out.println("Received event: " + p1.getType());
    // if the arrival event is the one that client subscribes,
    // the EventListner can trigger the call back function in the
client
    if (p1.getType().equalsIgnoreCase(m_eventType)) {
         m_client.executeCallBack(p1);
    }
   }
}
```

Compile and Run the Java Client

To compile the Java client, use the following command:

```
set JAVA_HOME = <the path of JDK>
set OneWorld_HOME = <the path of ERP installation>
set CLASSPATH=%OneWorld_HOME%\system\classes\Kernel.jar
set
CLASSPATH=%CLASSPATH%;%OneWorld_HOME%\system\classes\Connector.jar
set CLASSPATH=%CLASSPATH%;%OneWorld_Home%\system\classes\log4j.jar
set CLASSPATH=%CLASSPATH%;%OneWorld_HOME%\system\classes\xalan.jar
set CLASSPATH=%CLASSPATH%;%OneWorld_HOME%\system\classes\xerces.jar
%JAVA_HOME%\bin\javac -classpath %CLASSPATH% EventClient.java
EventListenerImpl
```

To run the Java client, use the following command:

```
set JAVA_HOME = <the path of JDK>
set OneWorld_HOME = <the path of ERP installation>
set CLASSPATH=%OneWorld_HOME%\system\classes\Kernel.jar
set
CLASSPATH=%CLASSPATH%;%OneWorld_HOME%\system\classes\Connector.jar
set CLASSPATH=%CLASSPATH%;%OneWorld_Home%\system\classes\log4j.jar
set CLASSPATH=%CLASSPATH%;%OneWorld_HOME%\system\classes\xalan.jar
set CLASSPATH=%CLASSPATH%;%OneWorld_HOME%\system\classes\xerces.jar
%JAVA_HOME%\bin\java -classpath %cp% -Dconfig_file=.\jdeinterop.ini
EventClient
```

Java Connector Architecture Resource Adapter

The ERP Java connector architecture (JCA) resource adapter allows Java2 Platform, Enterprise Edition (J2EE) components to use a standard interface to connect to the ERP system. A resource adapter is a system-level software driver that enables J2EE components to communicate with a back-end enterprise information system (EIS) through a JCA-compliant application server when a resource adapter for the specific EIS is deployed to the server. J2EE components consist of Servlets, JavaServer Pages (JSPs), and Enterprise JavaBeans (EJBs).

J2EE components and applications built with J2EE components can execute ERP business functions through the ERP JCA resource adapter. ERP business functions are accessed through the JCA standard client interface, the Common Client Interface (CCI). The ERP JCA resource adapter is fully compliant to the Java2 Platform, Enterprise Edition (J2EE) JCA 1.0 Specification and should work with any application server that is J2EE 1.3 certified.

Note

Some application servers are known to not be J2EE 1.3 certified but support some J2EE 1.3 features, including JCA 1.0. Check with your application server vendor to determine whether your application server supports JCA 1.0.

See Also

- *J2EE Connector Architecture* (*http://java.sun.com/j2ee/connector/*) for more information about J2EE Connector architecture

JCA 1.0 Specification Optional Features

The JCA 1.0 Specification identifies optional features for developing a resource adapter. The following table addresses the level-of-support that the ERP JCA resource adapter provides for the optional features identified in the JCA 1.0 Specification.

Transactions	The ERP JCA resource adapter is classified as a LocalTransaction resource adapter. The ERP JCA resource adapter allows either no transactions during business function calls, or transactions local to ERP during those same calls (local transaction). The resource adapter does not support XA Transactions, which are transactions that span multiple Enterprise Information Systems (EISs).
Client Interface	The ERP JCA resource adapter supports the optional common client interface (CCI), which is modeled after the Java Database Connectivity (JDBC) client API. This relatively simple Java API should significantly reduce the learning curve for using the ERP JCA resource adapter.

Reauthentication	The ERP JCA resource adapter does not support the switching of a set of ERP user credentials on an existing ERP user session. User credentials is usually a concern of the application server and should not affect client development.
Input/Output Records	The ERP JCA resource adapter supports the MappedRecord interface, which is a data type of key-value pairs. The MappedRecord interface is further discussed in the Input/Output Data section of this document. The CCI interfaces IndexedRecord and ResultSet are not supported as they are not relevant to the type of output from ERP business functions.
Authentication	The ERP JCA resource adapter supports BasicPassword authentication, which indicates to the application server how to handle container-managed signon. The resource adapter does not support any other form of authentication, such as Kerberos authentication through the GenericCredential interface. The Signon Types section of this document provides more information about authentication with the resource adapter.
ManagedConnectionFactory Properties	The JCA Specification identifies the following properties as standard; however, these properties are optional properties for the ManagedConnectionFactory class, which is the main class configured with ERP-specific properties during deployment of the resource adapter: • ServerName • PortNumber • UserName • Password • ConnectionURL The J.D. Edwards JCA resource adapter does not support the above list of properties, as these properties are either irrelevant properties or are configured elsewhere in the resource adapter. The Deployment Settings section of this document addresses other properties that are defined by the ERP JCA resource adapter. **Note** The deployment tool of your particular J2EE application server might list the above properties as configurable for the resource adapter. The ERP JCA resource adapter will not use values that you assign to these properties.

Number of Deployed Resource Adapters	The JCA Specification allows for the possibility of deploying the same resource adapter multiple times on a given application server. This provides for potential connectivity to multiple versions of the same EIS for a one resource adapter-to-many-EIS version ratio. The ERP JCA resource adapter supports the deployment of only one ERP JCA resource adapter per application server (essentially one resource adapter per virtual machine). **Note** You can install different JCA resource adapters (those other than for ERP) on the same application server.
Non-Managed Scenario	The ERP JCA resource adapter must be used with an application server or an application client. If you want to access ERP business functions through Java outside of an application server or application client, you should use the Java connector directly.

See Also

- The online *JCA Javadoc* (*http://java.sun.com/j2ee/apidocs-1_0-fr/api/index.html*) for official online JCA Java documentation (APIs)

Assembly and Components

The packaging of a resource adapter is defined in the JCA 1.0 Specification. However, because some application servers require additions to the standard Resource Adapter Archive (RAR) file, it is not possible to distribute a single RAR file that can be deployed to all application servers. The components that you need to assemble for the ERP JCA resource adapter are identified below. Consult your application server documentation for instructions on how to use the assembly tool and to understand what additional components might be required for a resource adapter to be operational with your application server. Typically, an additional deployment descriptor is required. Additional information required by an application server is usually for performance tuning and for configuration settings.

Components

A RAR file is a file that is in Java Archive (JAR) File Format with a .rar extension instead of a .jar extension. The file structure for a RAR file is:

- /META-INF/ra.xml
- /<all necessary JAR files>

The ra.xml file is the standard resource adapter deployment descriptor and must be put in the META-INF directory of the RAR file. The ra.xml file must be named *exactly* ra.xml. The ra.xml file for the ERP JCA resource adapter is provided in the system/classes/samples directory on the ERP CD. All other JAR files go in the root directory of the RAR file. The JAR files are provided in the system/classes directory on the ERP CD. The required resource adapter JAR files include:

- owra.jar
- connector.jar
- database.jar
- Kernel.jar
- log4j.jar
- xalan.jar
- xerces.jar

Note

Only use the versions of these JAR files that come with your ERP distribution.

When your RAR file is finally created, the META-INF directory of your RAR file might contain a Manifest.mf file. The Java JAR tool usually creates the Manifest.mf file automatically. The Manifest.mf file complies with the JAR file format, and it is acceptable for the Manifest.mf file to be in the RAR file.

Deployment and Configuration

The methods and tools for configuring and deploying a resource adapter vary between application servers and even between versions of the same application server. Consult your application server's documentation for information about how to configure and deploy a resource adapter.

Additional settings that are required for the ERP JCA resource adapter to be deployed and to operate correctly include the following:

- Security permissions
- jdeinterop.ini settings
- CLASSPATH settings
- Configurable properties
- JNDI settings

Note

Only one ERP JCA resource adapter can be deployed per application server.

Security Permissions

The JCA 1.0 Specification defines the standard Java security permissions that must be granted to all resource adapters by an application server. The ERP JCA resource adapter needs additional security permissions in order to operate. These permissions are listed in the deployment descriptor (ra.xml file). Most application servers dynamically grant these permissions to the resource adapter during deployment. Some application servers have other methods of granting the resource adapter additional permissions, including modifying a Java security policy file, which might require that you restart your application server to take effect.

If your application server does not dynamically grant the security permissions to a resource adapter based on the contents of the deployment descriptor, you will need to grant the resource adapter the permissions listed in the security-permission-spec elements of the deployment descriptor.

jdeinterop.ini Settings

Because the resource adapter is built on top of the Java connector, it is necessary to configure the appropriate settings in your jdeinterop.ini file to make the Java connector operational. The resource adapter introduces no new settings into the jdeinterop.ini file.

jdelog.properties Settings

The jdelog.properties file, which controls logging for the ERP JCA resource adapter, is preconfigured with sample values for each element. You need to change the file locations within the properties file to reflect your specific directory and file structure. The properties file includes comments that indicate how to enable more specific logging for debugging purposes. The default configuration in the jdelog.properties file sends all ERROR and FATAL messages to one file and sends all DEBUG, INFO, WARN, ERROR, and FATAL messages to another file.

The JCA 1.0 Specification allows for resource adapter-specific logging messages to be sent to a separate log file, which can be configured according to your application server (see your application server documentation). The messages that are sent to this log file will be redundant to and are a subset of the messages that are sent to the log file defined in the jdelog.properties file. This redundancy is an intentional ERP JCA resource adapter design decision for the following reasons:

- The JCA logging mechanism does not provide a method for logging messages from the connector on which the resource adapter is built. The logging properties file allows all logging messages from the connector as well as the resource adapter to be logged in a central location.
- The JCA logging mechanism potentially can be enabled or disabled by the application server without the user's control. It was desired to put a logging mechanism in place that the user could specifically configure and control.

CLASSPATH Settings

The ERP JCA resource adapter requires that the complete path to your jdelog.properties file be placed in your server's CLASSPATH. This path cannot include the name of the file, and the path must end with a slash, which designates that the last item in the path is a directory and not a file. The name of the properties file is required to be *jdelog.properties*. The logging mechanism will look for the logging properties file in all directories in the CLASSPATH.

The JDBC driver for your ERP database needs to be in your server's CLASSPATH so that the proper database connections can be made.

Note

Some servers require all of the JAR files within the resource adapter RAR file to be placed in the server's CLASSPATH. If you encounter a *NoClassDefFoundError* while running a Web application that is using the resource adapter, try putting all of these JAR files in the server's CLASSPATH and restarting the server. Consult your server documentation for further ClassLoader issues.

Configurable Properties

The ERP JCA resource adapter deployment descriptor (ra.xml file) contains properties that must be assigned values specific to your environment. The following table identifies the configurable properties and a description of the information required.

configFile	The location, including the path and filename, of the jdeinterop.ini file. You must assign a value to this property.
owVersion	The version of ERP to which the resource adapter will connect, such as B9, SP1. This property is for display purposes only and can contain any value. The value you enter in this property is not validated against your ERP installation. You must assign a value to this property.
defaultEnvironment	It is possible in a resource adapter web application to map a user's web credentials to a set of ERP user credentials. This mapping, which is called container-managed sign-on, prevents the user from having to present different credentials multiple times while using a single web application. Container-managed sign-on maps a given username and password to an ERP username and password. Container-managed signon mapping is specific to each application server. For ERP, the defaultEnvironment property is used to add a valid ERP environment to the username and password mapped by the application server, which allows for proper ERP signon. The defaultEnvironment property is only used during container-managed signon. If you use container-managed signon, you must assign a value to this property.
defaultRole	In addition to username, password, and environment, ERP signon requires a role. The DefaultRole property has a default value of **ALL*, which allows the user to assume all valid roles for the ERP username. You do not need to assign a value for defaultRole if this is the value you want to use. The defaultRole property is used during container-managed signon only.

Consult the documentation for your application server to determine if other deployment settings are required. For example, your application server might require setting the JNDI name of the resource adapter (which must be established at some point in the deployment process) or connection pooling parameters.

Java Naming and Directory Interface Settings

For communication between your web application and the ERP JCA resource adapter, the web application must perform a Java Naming and Directory Interface (JNDI) lookup of the ConnectionFactory of the resource adapter. The web application obtains an ERP connection and interacts with ERP through the ConnectionFactory. The method of assigning a JNDI name to the ConnectionFactory for the ERP JCA resource adapter is specific to and documented by your application server.

Note

A specific requirement of the ERP JCA resource adapter is that only one ConnectionFactory can be established per application server. Your application server might not restrict you to this requirement.

The JNDI is part of the Java Enterprise API set that provides Java technology-enabled applications with a unified interface to multiple naming and directory services in the enterprise.

When you configure a ConnectionFactory, you are likely going to be provided the method for assigning values to the Configurable Properties.

Common Client Interface

The Common Client Interface (CCI) is the JCA-recommended client API for all resource adapters. The ERP JCA resource adapter provides an implementation of CCI as the client interface. The following code example along with subsequent paragraphs show how to use the CCI API.

Example: Common Client Interface Code Sample

The following example code shows how to implement a CCI for the ERP JCA resource adapter. In the example, the elements in quotes have descriptive names and must have values valid to your environment in a real Java class. The line numbers in the example code are not part of the code but are for reference in subsequent paragraphs.

```
import com.jdedwards.system.connector..dynamic.jcaplugin.ImageBSFNInteractionSpecImpl;
import com.jdedwards.system.jca.cci.ConnectionSpecImpl;

import javax.naming.Context;
import javax.naming.InitialContext;
import javax.naming.NamingException;
import javax.resource.ResourceException;
import javax.resource.cci.Connection;
import javax.resource.cci.ConnectionFactory;
import javax.resource.cci.Interaction;
import javax.resource.cci.MappedRecord;
import javax.resource.cci.RecordFactory;
import javax.resource.cci.ResourceWarning;

public class SomeClass {
```

```
public void someMethod() {

    try {
      // get the naming context
      Context nc = new InitialContext();

      // lookup the connection factory
      ConnectionFactory conFact = (ConnectionFactory)nc.lookup("Resource Adapter JNDI
Name");

      // create a ConnectionSpec
1.    ConnectionSpecImpl conSpec = new ConnectionSpecImpl("username", "password",
"environment", "role");

      // get the Connection to ERP
2.    Connection con = conFact.getConnection(conSpec);

      // create an Interaction
      Interaction ix = con.createInteraction();

      // create and populate the InteractionSpec
      OWBSFNInteractionSpecImpl ixSpec = new OWBSFNInteractionSpecImpl();
      ixSpec.setBusinessFunction("ERP Business Function Name");

      // get a RecordFactory
      RecordFactory rf = conFact.getRecordFactory();

      // create the input MappedRecord
3.    MappedRecord inputRecord = rf.createMappedRecord("any descriptive name");

      // populate the input MappedRecord with the input values
4.    inputRecord.put("ERP Business Function Parameter Name",
                      "ERP Business Function Parameter Value");

      // execute the ERP Business Function, putting the results in the output
      // MappedRecord
5.    MappedRecord outputRecord = (MappedRecord)ix.execute(ixSpec, inputRecord);

      // get results
      Object value = outputRecord.get("ERP Business Function Parameter Name");

      // get ERP Business Function warnings, if any
      ResourceWarning warning = ix.getWarnings();

      // close the Interaction
      ix.close();

      // close the Connection
      con.close();
    } catch (ResourceException e) {
      // handle resource adapter-related Exceptions here
    } catch (NamingException e) {
      // handle JNDI-related Exceptions here
    }
```

```
  }
}
```

Signon Types

The ERP JCA resource adapter provides the following types of ERP signons:

- Container-managed signon
- Component-managed signon

Container-managed signon

When container-managed signon is used, the application server maps a web application user to a given ERP user. In this case, ERP user credentials are not provided in the CCI code. If you use container-managed signon, line 1 of the example code would not exist, as you do not need to create an instance of the ConnectionSpecImpl class. Line 2 of the example code would be changed to the following:

```
Connection con = conFact.getConnection();
```

Component-managed signon

When component-managed signon is used, the code provides specific ERP credentials (either through coding specific credentials or by obtaining ERP credentials through user entry in the web application) to the ERP JCA resource adapter for ERP signon. In the example code, lines 1 and 2 illustrate component-managed signon. As shown in line 1 of the example code, an instance of the `ConnectionSpecImpl` class is first created with the ERP user credentials. That instance is then passed to the getConnection method, which is shown in line 2.

Component-managed signon is also known as application-managed signon.

See Also

- *Configurable Properties* in the *ERP JCA Resource Adapter* section of the *Connectors Guide* for more information about container-managed signon.

Subclasses

The import statements at the top of the example code show that most of the classes that you use to interact with the ERP JCA resource adapter are JCA classes (those classes in the javax.resource package and sub-packages) and not J.D. Edwards-specific implementations of JCA interfaces. J.D. Edwards provides the following implementation classes:

- ConnectionSpecImpl
- xxxxInteractionSpecImpl

The ConnectionSpecImpl class supplies the required ERP user credentials to the getConnection method. The ConnectionSpecImpl class is one of the signon types. Line 1 in the example code shows how to use the ConnectionSpecImpl class.

The purpose of the xxxxInteractionspecImpl class is to establish the necessary ERP business function information prior to execution in ERP. The xxxxInteractionSpecImpl classes vary, depending on the type of ERP business function spec source. The ERP business function spec source is a file or location that describes an ERP business function. Each implementation class, which is a concrete class of the javax.resource.cci.InteractionSpec interface, include methods that set values. These “setter” methods must be called and given valid values prior to executing the ERP business function through the resource adapter.

ImageBSFNInteractionSpecImpl

The ImageBSFNInteractionSpecImpl implementation class gets the ERP business function spec from an XML image file, which must be generated beforehand.

Class: com.jdedwards.system.connector.dynamic.jcaplugin.ImageBSFNInteractionSpecImpl

Method: setBusinessFunction(String value)

> Sets the *exact* name of the ERP business function.

Method: setImageFilename(String value)

> Sets the complete path and filename of the dynamic Java connector ERP spec image that contains the definition of the corresponding ERP business function.

OWBSFNInteractionSpecImpl

The OWBSFNInteractionSpecImpl implementation class gets the ERP business function spec directly from a call to ERP. This method might take a little longer to execute a business function the first time the business function is called. The business function is stored in memory, and execution should be quicker in subsequent calls.

Class: com.jdedwards.system.connector.dynamic.jcaplugin.OWBSFNInteractionSpecImpl

Method: setBusinessFunction(String value)

> Sets the *exact* name of the ERP business function.

Input and Output Data

The MappedRecordImpl class handles both sending input data to the resource adapter and receiving the output data that is the result of executing the business function. Lines 3 and 4 of the example code illustrate inputting data, and line 5 illustrates obtaining the output data. A MappedRecord is a correlation of key/value pairs. The key represents the exact business function parameter name, and value defines the key.

Input data for values can be supplied in one of the following ways:

- Use a String
- Use a native Java data type

The ERP JCA resource adapter examines the input data on a parameter-by-parameter basis. If the input data type is String, the resource adapter will attempt to convert the input data to the appropriate Java data type for the specified parameter. If both the actual parameter type and the input data are String, the resource adapter will pass the input data through unchanged. If the input parameter is a native Java data type, the resource adapter will pass the input data through unchanged.

If the native Java data type is incorrect or if the parameter name is invalid for the given business function, the resource adapter will throw an Exception.

The following table lists the ERP business function types and their corresponding native Java data type:

ERP Data Type	Native Java Data Type
ID	java.lang.Integer
char (length of only 1)	java.lang.Character
JDEDATE	java.util.Date
Calendar	com.jdedwards.base.datatypes.JDECalendar (located in Kernel.jar file)
MATH_NUMERIC	com.jdedwards.system.lib.MathNumericImpl (located in Kernel.jar file)
char (variable length)	java.lang.String

The output of all business functions will result in the data in the MappedRecordImpl being in the native Java data types. If you prefer deal with only String-formatted output, you can make the following call on the output MappedRecordImpl for each parameter retrieved:

```
String value = outputRecord.get("parameter name").toString();
```

Logging

Message logging for the ERP JCA resource adapter is controlled by the jdelog.properties file, which was previously discussed.

Exceptions

The parent Exception class for all exceptions thrown by the ERP JCA resource adapter is javax.resource.ResourceException.

See Also

- *JCA Javadoc* (*http://java.sun.com/j2ee/apidocs-1_0-fr/api/index.html*) for the complete set of Exception subclasses and the methods that throw each Exception

Samples

The samples that are supplied with the resource adapter show how to use the resource adapter's API as well as the JCA API in general, and to demonstrate the functionality of the resource adapter. Address Book Query, Sales Order Entry, and Purchase Order Entry are included samples. The source code is delivered in your ERP distribution.

The following discussion addresses a limitation of the architecture and a potential solution.

Each sample is a group of Servlets, where one Servlet obtains a connection to ERP and then places that connection object in the HttpSession. The connection object is then used by succeeding Servlets to perform additional ERP business function calls. The sales order and purchase order examples use BeginDoc, EditLine(s), and EndDoc business function calls. All of these business function calls must be made over the same connection, because that is associated with a single user session in ERP. Because these Servlets are used by the end user in a Web browser, it is possible for the end user to click the Back and Forward buttons, potentially causing the business functions to be invoked in a sequence that is improper according to the business rules in ERP.

A potential solution for ensuring business functions are invoked in the proper sequence is to use a wrapper. When you develop your application, wrap all of the business function calls into a single enterprise JavaBean (EJB). The EJB stores the BeginDoc, EditLine(s), and EndDoc information and only connects to ERP and submits the information when the Web user completes all entry information (for example, a final Submit page). If the Web user clicks the Back button on the browser and resubmits a BeginDoc before the final Submit, only the BeginDoc object in the EJB is updated.

Architecture and Installation

Most Servlets have a two-character class name prefix that indicates which sample they represent, for example AB: Address Book, SO: Sales Order, PO: PurchaseOrder. Each group has an xxLogin Servlet as well as several other Servlets in the same package, each of which displays a separate Web page for user input or for business function output. The packages for the samples are in the “com.jdedwards.system.connector.dynamic.jcaplugin.samples…” package and subpackages.

The root *samples* package contains the Disconnect Servlet, which is common to all of the applications. The root *samples* package also contains utility classes that are needed by all of the Servlets. These utility classes are not Servlets and should not be declared as Web components in your Web application.

The packaging and deployment of a Web application to an application server is different for all application servers. Generally, you will need to package a standalone Web application where the Web components are the Disconnect Servlet and all the Servlets in the subpackages. The subpackages are addressbook, salesorder, and purchaseorder. You can package and deploy the samples independently.

Troubleshooting Checklist

- The directory location of your jdelog.properties file must be in your server’s CLASSPATH. For example, if your jdelog.properties file is in the following location:

  C:\JCA\logs\jdelog.properties

  you must have the following entry in your server’s CLASSPATH:

  C:/JCA/logs/

  Be sure to include a slash at the end of the path to indicate that *logs* is a directory and not a file. When you make a change to your server’s CLASSPATH, you must restart your server.

- Some servers read the <security-permission-spec> element of the resource adapter's deployment descriptor (the ra.xml file) and dynamically grant the resource adapter the security permissions listed in those elements. If you are executing a resource adapter-based application and experience a java.xxx.xxxPermission Exception, you will have to manually add the contents of the <security-permission-spec> elements to your server's policy file. Consult your server's documentation for the location and format for editing the policy file. You should be able to simply copy and paste the elements into the server's policy file. Any changes to the policy generally require a server restart to take effect.

  If you make the changes and still experience Permission Exceptions, you might need to move some of the permission elements that you copied from the *resource adapter* domain in the policy file to the *default domain* in the policy file. This is because the resource adapter classes, especially if present in the server's CLASSPATH, might reside in the *default domain* and not the *resource adapter* domain.

jdeinterop.ini

The jdeinterop.ini file includes settings the server might need. The default location for the file is `c:\`; however, this location is configurable. This section details the settings found in the jdeinterop.ini file. Information is organized by section, for example [JDENET].

[OCM]

The following settings are used only by the COM connector.

Setting	Typical Value	Purpose
DSN=	ODA ITTND17	The data source name from the system DSN of your ODBC setting
OCM Datasource=	COM OCM	System data source for ERP 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 AS400, the separator is /

The following setting is used only by the dynamic Java connector.

Setting	Typical Value	Purpose
OCMEnabled=	True	Turn on or turn off OCM inside the Java connector. A value of *true* indicates turned on.

[CACHE]

The following settings are used only by the Java interoperability connectors.

Setting	Typical Value	Purpose
UserSession=	0	Timeout value (in milliseconds) for the dynamic Java connector user session. A zero (0) indicates infinite timeout.
SpecExpire=	30000000	Maximum time (in milliseconds) that dynamic Java connector will keep the fetched spec in the cache.

[JDENET]

Setting	Typical Value	Purpose
enterpriseServerTimeout=	90000	Timeout value for a request to the enterprise server
maxPoolSize=	30	JDENET socket connection pool size
serviceNameConnect=	6004	Port number used by the ERP security server. This setting is used only by the Java connector.

[SERVER]

Setting	Typical Value	Purpose
glossaryTextServer=	JDED:6010	The 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]

Setting	Typical Value	Purpose
NumServers=	1	Number of security servers set
SecurityServer=	JDED	The ERP security server. This setting is used only by the Java connector.

[DEBUG]

The [DEBUG] section is used only by the COM connector.

Setting	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 tracing on by default. 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 tracing on by default. However, tracing is useful for debugging.

[INTEROP]

Setting	Typical Value	Purpose
enterpriseServer=	JDED	The ERP enterprise server
port=	6010	The port number of the ERP Enterprise server
manual_timout=	300000	The timeout value for a transaction in manual commit mode
Repository	c:\PeopleSoft\ Interop\repository	Points to the location of the repository directory containing business object libraries (generated JAR files)

[EVENTS]

Setting	Typical Value	Purpose
port=	6002	The socket port number where the EventListener will receive the events from the enterprise 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 will cause 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 enterprise server.

[JDBj-BOOTSTRAP SESSION]

The JDBj sections are used only by the Java interoperability connectors. The JDBj sections define the OCM bootstrap data source for accessing the OCM bootstrap tables. These sections also define database connection configuration needed by JDBj (ERP Database implementation). Settings could vary depending on the type of database.

The [JDBj-BOOTSTRAP SESSION] section defines the bootstrap session. Bootstrap session information is used for sign on to provide access to system tables. JDBj logs you on through the security server at bootstrap time. If the session information is not valid, you will not be able to access the ERP database.

Setting	Typical Value	Purpose
User=	JDE	User name for database access
Password=	JDE	Password for database access
Environment=	PDEV	The ERP environment
Role=	*ALL	The ERP user role

[JDBj-BOOTSTRAP DATA SOURCE]

The JDBj sections are used by the Java interoperability connectors only. The JDBj sections define the OCM bootstrap data source for accessing the OCM bootstrap tables. These sections also define database connection configuration needed by JDBj (ERP Database implementation). Settings could vary depending on the type of database.

The [JDBj-BOOTSTRAP DATA SOURCE] section defines the physical bootstrap data source information. Bootstrap data source information defines the data source where the OCM resides. JDBj uses this at bootstrap time and later to look up OCM entries on demand. If the data source information is not valid, you will not be able to access the ERP database.

Note

The information in the JDBj-BOOTSTRAP DATA SOURCE] section should match the corresponding information specified in the [DB SYSTEM SETTINGS] section of your installation's JAS.INI and JDE.INI files.

Not all of the settings are required for all platforms. The following table provides and explanation of the settings.

Setting	Typical Value	Purpose
name=	System – B9	Data source name
server=	JDE	Database server name
DatabaseType=		Database type. DatabaseType depends on the platform: • AS/400 - databaseType=I • Oracle – databaseType=O • SQL Server – databaseType=S • UDB – databaseType=W
database=		Database
serverPort=		Database server port
physicalDatabase=		Physical database name
owner=	SystemB9	Database owner

Not all settings are needed for all platforms. The following table identifies the required settings by platform:

Platform	Required Settings
AS/400	name, databaseType, server, physicalDatabase
Oracle	name, databaseType, database, owner
SQL Server	name, databaseType, server, serverPort, physicalDatabase, owner
UDB	name, databaseType, database, owner

[JDBj-CONNECTION POOL]

The JDBj sections are used only by the Java interoperability connectors. The JDBj sections define the OCM bootstrap data source for accessing the OCM bootstrap tables. These sections also define database connection configuration needed by JDBj (ERP Database implementation). Settings could vary depending on the type of database.

The [JDBj-CONNECTION POOL] section defines the connection pool settings. These settings are used only when J2EE connection pooling is not available.

Setting	Typical Value	Purpose
jdbcPooling=	False	Allows JDBC pooling
minConnection=	5	Minimum physical connection in the connection pool
maxConnection=	5	Maximum physical connection in the connection pool
initialConnection=	5	Initial connection when started up
poolGrowth=	5	The connection size to increase when pool size grows
connectionTimeout=	1800000	The timeout value, in milliseconds, for a connection to be made
cleanPoolInterval=	600000	Interval time when connection pool is cleaned

[JDBj-JDBC DRIVERS]

The JDBj sections are used only by the Java interoperability connectors. The JDBj sections define the OCM bootstrap data source for accessing the OCM bootstrap tables. These sections also define database connection configuration needed by JDBj (ERP Database implementation). Settings could vary depending on the type of database.

The [JDBj-JDBC DRIVERS] section defines the JDBC drivers to be used by the Java interoperability connectors. You only need to set the setting that is applicable for your database type.

Setting	Typical Value	Purpose
ORACLE=	Oracle.jdbc.driver.OracleDriver	JDBC driver for Oracle
AS400=	Com.ibm.as400.access.AS400JDBCDriver	JDBC driver for AS400
SQLSERVER=	Com.Microsoft.jdbc.sqlserver.SQLServerDriver	JDBC driver for MS SQL Server
UDB=	COM.ibm.DB2.jdbc.app.DB2Driver	JDBC driver for UDB

[JDBj-ORACLE]

The JDBj sections are used by the Java interoperability connectors only. The JDBj sections define the OCM bootstrap data source for accessing the OCM bootstrap tables. These sections also define database connection configuration needed by JDBj (ERP Database implementation). Settings could vary depending on the type of database.

The [JDBj-ORACLE] section defines Oracle tnsnames location. You only need to set this setting when your database type is Oracle.

Setting	Typical Value	Purpose
tns=	C:\Oracle\Ora901\network\ADMIN\tsnames.ora	Oracle tnsnames location

jdeLog.properties

The logging utility in the dynamic Java connector, the Java connector, and Java connector Architecture (JCA) is built on top of Apache open source project Log4j. The jdeLog.properties file defines the settings for the logging configuration. The jdeLog.properties file should be physically located in CLASSPATH.

The following settings provide a sample configuration file for root configuration:

- Root configuration

  jdelog.rootLogger=DEBUG,JDELOG,JASLOG

  jdelog.loggerFactory=com.jdedwards.base.logging.log4j.JdeLoggerFactory

  jdelog.reloadInterval=60

- File handler for root log

  jdelog.handler.JDELOG=com.jdedwards.base.logging.log4j.FileHandler

  jdelog.handler.JDELOG.File=\\jderoot.log

  jdelog.handler.JDELOG.Level=ERROR

  jdelog.handler.JDELOG.Append=TRUE

  jdelog.handler.JDELOG.MaxBackupIndex=1

  jdelog.handler.JDELOG.MaxFileSize=10MB

  jdelog.handler.JDELOG.format=com.jdedwards.base.logging.log4j.DefaultFormat

- File handler setting for jas log

  jdelog.handler.JASLOG=com.jdedwards.base.logging.log4j.FileHandler

  jdelog.handler.JASLOG.File=\\jas.log

  jdelog.handler.JASLOG.Level=ERROR

  jdelog.handler.JASLOG.Append=TRUE

  jdelog.handler.JASLOG.MaxBackupIndex=1

  jdelog.handler.JASLOG.MaxFileSize=10MB

  jdelog.handler.JASLOG.format=com.jdedwards.base.logging.log4j.DefaultFormat

- File handler setting for jasdebug log

  jdelog.Debug=DEBUG, jasdebug

  jdelog.handler.jasdebug=com.jdedwards.base.logging.log4j.FileHandler

  jdelog.handler.jasdebug.File=\\jasdebug.log

  jdelog.handler.jasdebug.Level=DEBUG

See Also

- *Log4j Project* (*http://jakarta.apache.org/log4j/docs/index.html*) on the Apache Jakarta Project Web site more information about the Log4j Project

iJDEScript

GenCOM and GenJava use a scripting language called iJDEScript that allows you to script code generation activities. Other than a few small differences, the scripting language is the same for these generators. You can use iJDEScript to:

- Rename business function libraries or select different business functions to create a custom interface; for example:

  library MyTestLibrary

  interface MytestInterface

  import B4200310 F4211FSEditLine

  import B000042

  This example selects the single business functions B4200310 F4211FSEditLine and B000042 for exposure.

- Use ERP object aliases for more meaningful names.
- Select ERP business functions to expose: for example:

  library MyAnotherLibrary

  importlib CAEC

  importlib CRUNTIME 1

  This example selects all of the business functions in the CAEC and CRUNTIME 1 libraries for exposure.

iJDEScript scripts have a simple syntax:

```
# comments begin with # and proceed to the end of line
# whitespace is ignored
login
importlib CAEC
build
```

iJDEScript Commands

iJDEScript supports a standard set of commands. These commands may vary slightly for GenCOM and GenJava. These variations are indicated in the command descriptions that follow.

Build Command

The build command tells the generator to generate code for all defined interfaces and build the appropriate libraries.

When the build command is complete, the interface definitions are released. Using the build command again only generates code for interfaces defined after the last build command.

Syntax

```
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

```
call sub [param [...]]
```

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

```
define name value
```

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

```
define name value
```

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

```
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 shown.

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 allows 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

```
import business-function [function [alias name]]
```

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 ERP 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 allow the user to restrict the import to one or more specific categories (1, 2, 3 and -; see the /Cat command line option).

Syntax

```
importlib library [category [...]]
```

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

```
interface interface [ProgID prog-id] [vi-prog-id]
```

COM 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

```
library name [parameterset]
```

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 ERP. If user, password, environment, and role are not specified, the user is prompted for the information.

Syntax

```
login [user password environment role]
```

Example

```
login me mypassword demo
```

Logout Command

The logout command tells the generator to log off ERP.

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

```
opt option value
```

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

```
rename interface new
rename interface method new
```

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 *message* on the console.

Syntax

```
say message
```

Example

```
say This is a test (%OwRelease%)
generate the output
This is a test (B733)
```

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

```
sub name
  commands
end
```

Example

```
sub GenerateLibrary
      define source %1%
      library JDECOMInterface%source%Cat1
  importlib %source% 1
# Create a library of all category 2 business functions in
source
      opt NoBSFN
      library JDECOMInterface%source%Cat2
```

```
  importlib %source% 2
      # Create a library of all 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 command in the shell.

Syntax

```
system command
```

Example

```
say This is a test
generates the output
This is a test
```