

PeopleSoft®

EnterpriseOne Xe
Interoperability
PeopleBook

September 2000

J.D. Edwards World Source Company
7601 Technology Way
Denver, CO 80237

Portions of this document were reproduced from material prepared by J.D. Edwards.

Copyright ©J.D. Edwards World Source Company, 2000

All Rights Reserved

SKU XeEAIO

J.D. Edwards is a registered trademark of J.D. Edwards & Company. The names of all other products and services of J.D. Edwards used herein are trademarks or registered trademarks of J.D. Edwards World Source Company.

All other product names used are trademarks or registered trademarks of their respective owners.

The information in this guide is confidential and a proprietary trade secret of J.D. Edwards World Source Company. It may not be copied, distributed, or disclosed without prior written permission. This guide is subject to change without notice and does not represent a commitment on the part of J.D. Edwards & Company and/or its subsidiaries. The software described in this guide is furnished under a license agreement and may be used or copied only in accordance with the terms of the agreement. J.D. Edwards World Source Company uses automatic software disabling routines to monitor the license agreement. For more details about these routines, please refer to the technical product documentation.

OneWorld Interoperability Guide

The OneWorld Interoperability guide consists of the following:

- Overview
- Connectors
 - COM
 - JAVA
 - CORBA
- XML
- APIs
 - OneWorld APIs Interoperability Models
 - Detailed Tasks for OneWorld APIs
 - Detailed Tasks for OneWorld Operations
 - Detailed Tasks for Custom Programming
 - Additional Information
- Appendix A: Interoperability Features Created by J.D. Edwards
- Appendix B: Interface Tables

Refer to the online Interoperability Interface Tables for more information.
- Appendix C: Business Function Documentation
- Appendix D: Open Database Access
- Appendix E: XML Format Examples



OneWorld Acronyms

The following is a list of acronyms that are commonly used in OneWorld and might appear in this guide.

API	Application Programming Interface
APPL	Application
BDA	Business View Design Aid
BSFN	Business Function
BSVW	Business View
COM	Component Object Model
CORBA	Common Object Request Broker Architecture
CRP	Conference Room Pilot
DBMS	Database Management System
DCOM	Distributed Component Object Model
DD	Data Dictionary
DLL	Dynamic Link Library
DS or DSTR	Data Structure
EDI	Electronic Data Interchange
ER	Event Rules
FDA	Form Design Aid
IDL	Interface Definition Language
NER	Named Event Rules

ODBC	Open Database Connectivity
OCM	Object Configuration Manager
OL	Object Librarian
QBE	Query by Example
RDA	Report Design Aid
SAR	Software Action Request
Specs	Specifications
SQL	Structured Query Language
TAM	Table Access Management
TBLE	Table
TC	Table Conversion
TDA	Table Design Aid
TER	Table Event Rules
UBE	Universal Batch Engine
WF	Workflow
XML	Extensible Markup Language

Table of Contents

Interoperability Overview	1-1
Benefits	1-1
OneWorld Interoperability Features	1-1
APIs	1-2
Middleware	1-3
OneWorld Database APIs	1-4
OneWorld Active Data Dictionary	1-4
OneWorld Business Functions	1-4
Other Industry Standard Support	1-5
Proprietary Support	1-5
Additional OneWorld Interoperability Options	1-5
Types of Interoperability	1-6
Batch	1-6
Interactive	1-6
Interoperability Solution Overview	1-7
Flat Files	1-8
EDI	1-9
OneWorld APIs	1-9
COM	1-10
CORBA	1-10
JAVA	1-10
XML	1-10
Table Conversion	1-10
Messaging Support	1-11
SAP ALE/IDoc	1-11
Open Data Access (ODA)	1-11
API Models	1-11

Connectors

Connectors	2-1
Choosing a Connector	2-1
Generating Business Function Wrappers	2-2
COM	3-1
Understanding the Component Object Model (COM)	3-3
OneWorld and COM	3-3
OneWorld COM Objects	3-3
OneWorld COM Interoperability Usage	3-4
OneWorld COM Server	3-6
COM Connector	3-6
Generated COM Components	3-7

COM Reliability	3-7
OneWorld GenCOM	3-8
OneWorld COM Server Deployment	3-9
Using the COM Generator (GenCOM)	3-11
Running GenCOM	3-11
Syntax	3-11
Options	3-12
ProgID	3-15
Using GenCOM Output	3-15
Visual Basic	3-15
Visual C++	3-17
Using BHVRCOM via COM	3-19
Installation Information	3-20
Setting Up a OneWorld Environment for GenCOM	3-21
Include Directories	3-21
Lib directories	3-22
MSDev Directories	3-22
Paths	3-22
Installing a COM Server on a Non-OneWorld Machine	3-25
Setting Up a OneWorld DCOM Server	3-27
Setting Up DCOM for a Server Environment	3-27
Setting Up Security on the COM Server	3-27
Setting Up DCOM for a Client Environment	3-31
Using the COM Wrapper Version Checker (CheckVer)	3-33
Running CheckVer	3-33
Syntax	3-33
Options	3-33
Using COM Tracing and Logging	3-35
Troubleshooting	3-35
Java	4-1
Understanding Java	4-3
Java and OneWorld	4-3
JDEDate	4-5
JDEMathNumeric	4-5
Understanding GenJava	4-6
Pure Java Component Usage	4-6
Using the Java Generator (GenJava)	4-7
Running GenJava	4-7
Syntax	4-7
Options	4-8
Using GenJava Output	4-10
Setting Up a OneWorld Client Environment for GenJAVA	4-15
PATH	4-15
CLASSPATH	4-15
Installing JAVA Components on a Non-OneWorld Machine	4-17
Understanding JAVA and CORBA Versioning	4-19
Migrating from Previous Releases	4-19
Static and Dynamic Modes	4-19

Using the JAVA Wrapper Version Checker (CheckVer)	4-21
Running CheckVer (GenJava)	4-21
Syntax	4-21
Example	4-21
Running CheckVer (GenCORBA)	4-22
Syntax	4-22
Example	4-22
CORBA	5-1
Understanding CORBA	5-3
CORBA and OneWorld	5-5
Understanding GenCORBA	5-6
Understanding the CORBA Generator (GenCORBA)	5-9
Running GenCORBA	5-9
Syntax	5-9
Options	5-10
Using GenCORBA Output	5-12
Setting Up a OneWorld Client Environment for GenCORBA	5-17
PATH	5-17
CLASSPATH	5-17
Setting Up an Environment for CORBA	5-19
Building and Deploying a CORBA Server	5-19
Setting up a CORBA Server	5-20
ijDEScript	6-1
ijDEScript Commands	6-2
jdeinterop.ini	7-1
[JDENET]	7-1
[SERVER]	7-1
[LOGS]	7-2
[DEBUG]	7-2
[INTEROP]	7-3
[CORBA]	7-4

XML

XML	8-1
Understanding XML	8-3
XML and OneWorld	8-3
ThinNet	8-4
Inbound and Outbound Synchronous XML CallObject	8-6
Inbound Synchronous Process Flow	8-6
Inbound Asynchronous XML Transactions	8-7
Outbound Asynchronous XML Transactions	8-7
Outbound Asynchronous Process Flow	8-8
Working with XML CallObject	8-9
Establish Session	8-9
Expire Session	8-9
Call Object	8-10

Explicit Transaction	8-10
Implicit Transaction	8-10
Prepare/Commit/Rollback	8-11
Terminate Session	8-12
Call Object Error Handling	8-12
BSFN Error Handling	8-12
Error Text	8-12
Multiple Requests per Document	8-13
On Error Handling	8-13
ID/IDREF Support	8-14
Return NULL Values	8-15
Enabling Outbound Z-Table Processes	8-16
Outbound Notification	8-16
XML Z-Table Inquiry API	8-18
XML Transaction Information Request	8-18
Example: Outbound Order Status XML Request & Response Format	8-18
Creating an XML Template	8-21
XML CallObject Templates	8-21
Setting the jde.ini File for XML	8-23

APIs

OneWorld APIs Interoperability Models	9-1
Processing Modes	9-1
Implementing Synchronous Transactions Into OneWorld	9-5
Detailed Tasks Using Native APIs	9-5
Implementing Asynchronous Transactions Into OneWorld	9-7
Detailed Tasks Using Native APIs	9-8
Detailed Tasks for OneWorld Operations	9-9
Implementing Batch Transactions Into OneWorld	9-11
Detailed Tasks Using Native APIs	9-12
Detailed Tasks Using OneWorld Operations	9-12
Additional Options	9-13
Implementing Synchronous Transactions From OneWorld	9-15
Detailed Tasks Using Native APIs	9-15
Implementing Asynchronous Transactions from OneWorld	9-17
Detailed Tasks Using OneWorld Operations	9-18
Detailed Tasks Using Custom Programming	9-19
Implementing Batch Transactions From OneWorld	9-21
Detailed Tasks Using OneWorld Operations	9-22
Detailed Tasks Using Native APIs	9-22
Additional Options	9-22
Detailed Tasks for OneWorld APIs	10-1
Connect to OneWorld	10-3
JDB_InitEnvOvr	10-3
JDB_InitUser	10-4
Example	10-5

Disconnect from OneWorld	10-7
JDB_FreeUser	10-7
JDB_FreeEnv	10-7
Example	10-8
Add Records to Interface Tables	10-9
JDB_OpenTable	10-9
JDB_InsertTable	10-10
JDB_CloseTable	10-10
Example	10-11
Retrieve Records from Interface Tables	10-13
JDB_OpenTable	10-13
JDB_SelectAll	10-14
JDB_Fetch	10-14
JDB_CloseTable	10-15
Example	10-16
Call OneWorld Business Functions	10-17
Variable Initialization	10-17
jdeCreateBusinessFunctionParms	10-18
jdeFreeBusinessFunctionParms	10-18
jdeErrorInitializeEx**	10-19
jdeErrorTerminateEx**	10-19
jdeAlloc	10-19
jdeFree	10-20
jdeCallObject	10-21
Application Error Retrieval	10-21
jdeErrorGetCountEx**	10-22
jdeErrorSetToFirstEx**	10-22
Example: Business Function Call	10-24
Call the OneWorld Completion Confirmation API	10-27
InteropOutboundConfirmationFunc	10-27
Example: Vendor Function	10-28
Place an Entry in the Subsystem Data Queue	10-29
Inbound Transaction Subsystem Data Structure	10-29
Requesting Inbound Transaction Confirmation	10-30
Example	10-31
Detailed Tasks for OneWorld Operations	11-1
Run the Input Batch Process	11-3
Run an Extraction Batch Process	11-7
Run a Subsystem Job from a OneWorld Menu	11-11
Enable Outbound Transaction Processing	11-13
Subscribe to Outbound Transactions	11-15
Using Data Export Controls	11-15
Processing Log Table	11-17
Check for Errors	11-19
Use a Revisions Application	11-21
Import from Flat Files	11-23
Setup Requirements for Flat File Conversion	11-24

Detailed Tasks for Custom Programming	12-1
Process Outbound Transactions	12-3
Vendor-Specific Outbound Batch Processes	12-3
Vendor-Specific Outbound Functions	12-3
Process Inbound Transaction Confirmation	12-5
Additional Information	13-1
Formatting Data	13-3
MATH_NUMERIC Data Type	13-3
JDEDATE Data Type	13-5
Using Scheduler	13-7

Appendices

Appendix A - Interoperability Features Created by J.D. Edwards	A-1
Transactions Into OneWorld	A-1
Inbound Through a Flat File	A-1
Inbound Transactions to Unedited Transaction Tables	A-2
Inbound Transactions Passed Directly to Master Business Functions	A-3
Transactions From OneWorld	A-3
Common Pieces of the Outbound Process	A-3
WorldSoftware Coexistence	A-5
Net Change Considerations	A-5
Information Structure	A-6
Creating Transactions Into and From OneWorld	A-7
Transaction Name	A-7
Unedited Transaction Tables	A-7
Revision Application	A-10
Purge Batch Process and Named Event Rules	A-10
Subsystem Business Function	A-10
Creating Transactions Into OneWorld	A-13
Creating an Inbound Transaction	A-13
Inbound Flat File	A-14
Creating Transactions from OneWorld	A-17
Task Summary for J.D. Edwards Interoperability Features	A-21
Appendix B - Interoperability Interface Table Information	B-1
Appendix C - Business Function Documentation	C-1
Business Function Documentation	C-3
Creating Business Function Documentation	C-3
Generating Business Function Documentation	C-9
Viewing Business Function Documentation	C-14
Appendix D - Open Data Access (ODA)	D-1
Hardware and Software Requirements	D-1
ODBC Component Files	D-2
Open Data Access Driver Architecture	D-2

Adding an ODA Data Source	D-5
Adding a File Data Source	D-6
Adding a System Data Source	D-7
Modifying a Data Source	D-8
Deleting a Data Source	D-9
Working with ODA	D-11
Example	D-13
ODA Error Messages	D-15
Appendix E – XML Format Examples	E-1
XML Format Examples (All Parameters)	E-3
XML Format Examples (All Parameters)	E-3
Inbound Sales Order XML Format (All Parameters)	E-3
Outbound Customer Create XML Format (all fields)	E-15
Outbound Customer Create XML Request and Response Format	E-15
XML Format Examples (Default Values)	E-19
XML Format Examples (Default Values)	E-19
Inbound Sales Order XML Format	E-19

Glossary

Index



Interoperability Overview

Interoperability is most often associated with software as a way to allow disparate software applications to work together. For example, interoperability makes it possible for a company to use applications from different vendors as if they were from a single vendor. Seamless sharing of function and information becomes possible.

Interoperability reduces or eliminates the problems of islands of automation. It allows business processes to flow from one application to another. Interoperability allows one system to work with another in a near real time fashion, sharing critical business information. Interoperability options become the “glue” between systems and applications.

Benefits

Interoperability offers the following benefits:

- Businesses can bring together applications and systems across an enterprise, irrespective of the vendors
- Collaborations can occur between trading partners to lower the cost of doing business or increase competitiveness
- Multiple systems can be linked together to share information in a real-time manner, delivering time-sensitive information to those who need it
- Disparate solutions encountered because of mergers or acquisitions can be quickly incorporated into the enterprise’s information technology solution

J.D. Edwards interoperability strategy includes wide-ranging APIs (both industry standard and OneWorld-based), object standards, interoperability processware solutions, and other methods.

OneWorld Interoperability Features

Full interoperability between different systems makes the flow of data between systems seamless to the user. J.D. Edwards provides a framework to mask the complexity of interoperability with external systems, and to simplify interfacing with third-party packages.

OneWorld’s interoperability strategy includes wide-ranging APIs (both industry standard and OneWorld based), object standards, interoperability engines, and other methods used to meet three important business objectives:

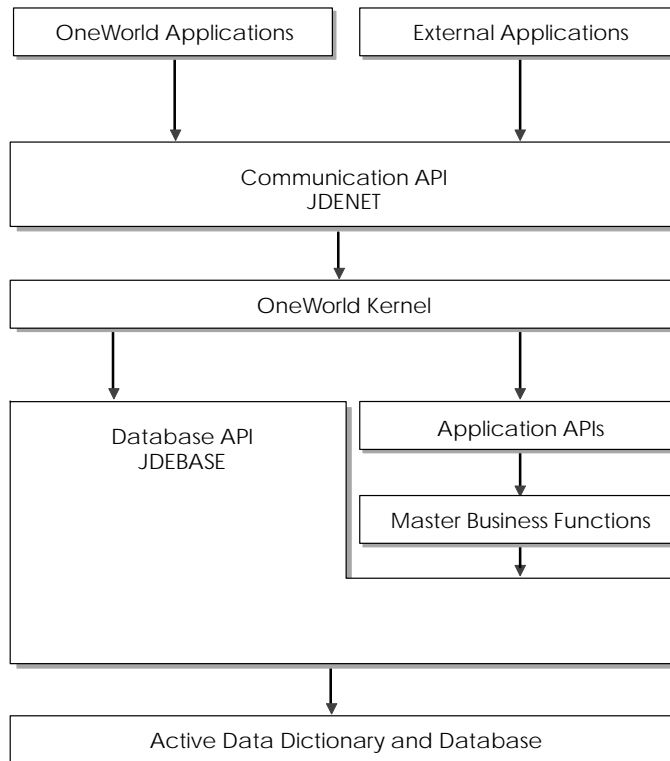


- Flexibility, Options, and Choice. J.D. Edwards gives you several possibilities in the types of applications and information you want to work with OneWorld - legacy, best-of-breed, customer management, reporting tools, and many more. The developer can make the right choice for your particular environment and needs.
- Investment Preservation. If you have existing applications you want to continue using, or if you have applications you are thinking about adopting, you can interface them to OneWorld. You can use industry standard methods if the existing or new technologies support them. Or, by using the same APIs that OneWorld uses internally, you gain all the benefits of this enterprise-ready architecture for the other applications you connect to OneWorld. Also, you will benefit from our ongoing upgrades and improvements to that architecture.
- Manageability. OneWorld is designed to make the interoperability process easily manageable.

APIs

OneWorld's component framework allows you to access environments and pass requests and their associated parameters. This framework includes business function wrappers that provide a single point of access to major and minor business functions. It also includes Master Business Function (MBF) wrappers that provide discrete Interface Definition Language (IDL) and parameter objects for each Master Business Function.

The following graphic illustrates how OneWorld and third-party applications can interact through the use of APIs and master business functions.



Middleware

In a client/server environment, applications must communicate across different platforms. These platforms can have different communication protocols, database management systems, and operating systems. For clients to talk to servers, and servers to talk to other servers, some mechanism must exist that can bridge multiple protocol and multiple vendor issues. This mechanism is a layer of software called middleware, which resides between the operating system and the business applications. It is important to have an application architecture that is based on a single, consistent middleware strategy.

J.D. Edwards provides the following types of middleware:

JDENet Communication Middleware	Performs connections from workstation to server and server to server, and sends messages for distributed requests. JDENet is a peer-to-peer, message-based, socket-based, multiprocess communications middleware solution.
--	--

JDEBase Database Middleware

Provides platform-independent APIs for multidatabase access. These APIs are used in two ways:

- By OneWorld applications that dynamically generate platform-specific SQL, depending on the data source request.
- As open APIs for writing advanced C business functions

JDEBase also provides workstation-to-server and server-to-server database access. To accomplish this, OneWorld is integrated with a variety of third-party database drivers.

OneWorld Database APIs

To access OneWorld data, you can use JDEBase, OneWorld's database middleware API that abstracts the underlying database APIs. JDEBase contains the same set of APIs that OneWorld developers use to build OneWorld applications.

These APIs allow you to develop applications without having to know and interpret the various versions of SQL used by different DBMSs. JDEBase translates requests for OneWorld data into database-specific Structured Query Language (SQL) statements. By using the JDEBase APIs, you can build a non-OneWorld program that can work with a OneWorld database.

OneWorld Active Data Dictionary

OneWorld has a repository of data item definitions called the Active Data Dictionary (ADD). It acts as the central repository of item definitions and contains such things as decimal placement and default values for table fields. The ADD assigns column and row descriptions to the various OneWorld tables, and stores context-sensitive help definitions.

When a third-party application reads or writes information into the OneWorld tables, the ADD must be accessed for attribute definition and verification. As such, J.D. Edwards provides access to the ADD to third-party applications.

OneWorld Business Functions

OneWorld business functions perform specific tasks, such as Journal Entry Transactions, Calculating Depreciation, and Sales Order Transactions. When performing synchronous calls to OneWorld, business functions are called to accomplish that task.

There are two types of business functions - regular business functions and master business functions (MBF). Regular business functions perform simple tasks, such as tax calculation or account number validation. Master business

functions perform more complex tasks, and may call several regular business functions to perform those tasks.

See the *OneWorld Development Tools Guide* for more information about OneWorld business functions.

Other Industry Standard Support

OneWorld supports industry standard interoperability functions, such as:

- Object Linking and Embedding (OLE) for exchange of different data types
- Dynamic Data Exchange (DDE) for static and dynamic links across applications
- SQL for storage and retrieval of data from and across relational databases
- Spec 1170 for portability across various operating systems
- Extended Messaging API (MAPI) for message exchange across differing mail and groupware applications
- Binary Large Object Type (BLOB) for media object attachments within OneWorld applications
- Transmission Control Protocol/Internet Protocol (TCP/IP) for data communication
- Java for Internet and intranet support
- Electronic Data Interchange for using the OneWorld Universal Batch Engine to fully integrate transaction-oriented files in three primary formats—X.12, ANSI, and EDIFACT.
- XML
- Microsoft COM
- IBM MQ-Series

Proprietary Support

OneWorld also supports some proprietary interoperability functions, such as:

- SAP IDOC

Additional OneWorld Interoperability Options

To offer additional interoperability options, OneWorld provides the following:

- Z Files. OneWorld includes support for importing and exporting Z data files (World or OneWorld tables that are used for batch posting of transactions) from external applications.

- Table Conversion. OneWorld has a table conversion utility you can use to gather, format, export, and import enterprise data. You can use Table Conversion for table-to-table conversions.

Types of Interoperability

J.D. Edwards provides both batch and interactive interoperability.

Batch

The following batch interoperability features are available:

- Electronic Data Interchange (EDI)
- Interoperability Interface Tables
- Table Conversion

These processes involve movement of bulk information.

Interactive

J.D. Edwards provides interactive support using the following technologies:

Native OneWorld APIs

APIs are routines that perform predefined tasks. OneWorld APIs make it easier for third-party applications to interact with OneWorld. OneWorld APIs are functions provided to manipulate OneWorld data types, provide common functionality, and access the database.

The published OneWorld APIs for OneWorld's communication middleware, allow third-party applications to access OneWorld functionality just as OneWorld applications do. These APIs provide the security and load balancing capabilities inherent in OneWorld.

Component Object Model (COM) and Distributed Component Object Model (DCOM)

COM is well-supported in the Win32 world. It is Microsoft's stated direction for components and application interoperability. COM is used to maximize client-side, or NT-server based interoperability.

COM plays an important role in providing logic and data sharing among disparate applications.

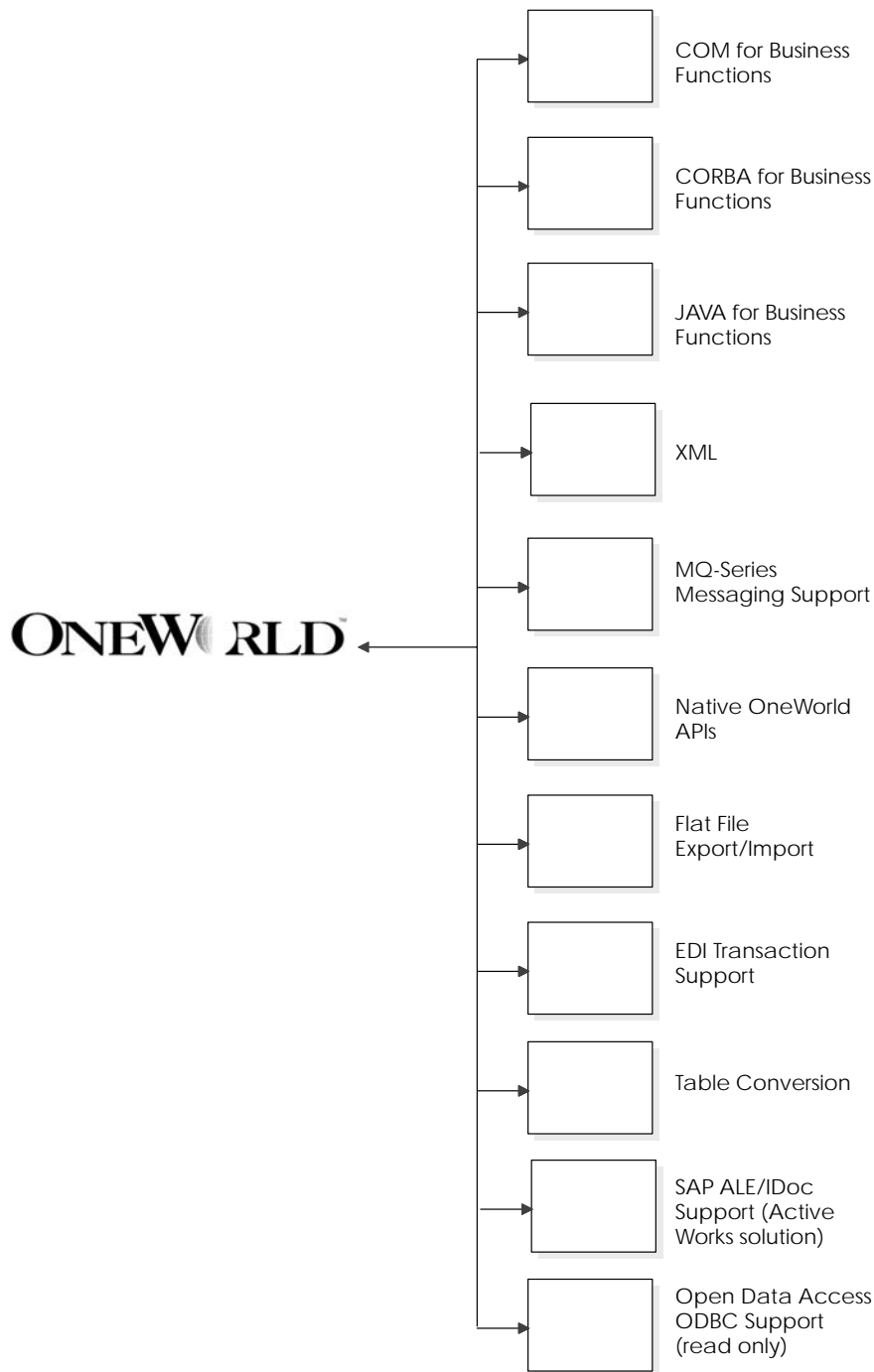
DCOM enables software components to communicate directly over multiple network transports, including Internet protocols such as HTTP.

JAVA	Java technology allows you to run the same application on a variety of different machines. You can also use security features with Java.
CORBA	CORBA is a standard defined by the Object Management Group (OMG). CORBA allows you to use objects independent of language or platform.
XML	XML is an extensible structured language that allows you to define how data is handled. It separates content from the format of the content so that information from one application can be used in a different application.
MQ-Series (Message queue services)	MQ-Series transports message-based application requests across more than 25 different platforms. Similar to OneWorld's JDENET middleware, MQ-Series handles message queuing, message delivery, and transaction monitoring. MQ-Series is a perfect complement to JDENET for sharing logic and information between OneWorld and third-party applications.
SAP ALE/IDoc/BAPI (Application Linking Embedding and Intermediate Document)	Application Linking and Embedding and IDoc (Intermediate Document) features give OneWorld the ability to share logic and data requests with SAP R/3. This capability is provided through Active Software's ActiveWorks, a solution that J.D. Edwards resells.

Together, these elements provide a comprehensive and cohesive interoperability solution through OneWorld.

Interoperability Solution Overview

The following diagram illustrates the interoperability solutions you can use with OneWorld.



For detailed information about available interoperability features for a specific application, refer to the guide for that specific application.

Flat Files

Flat files (also known as user-defined formats) do not have relationships defined for them like relational database tables do. Data in a flat file is stored as one continuous string of information. Flat files are usually text files stored on your

workstation or server, and typically use the ASCII character set. They are used to import or export data from applications that have no other means of interaction. For example, you might want to share information between OneWorld and another application. If the non-OneWorld application does not support the same databases that OneWorld supports, then flat files might be the only way to transfer data between the two applications.

EDI

Electronic Data Interchange (EDI) is the paperless, computer-to-computer exchange of business transactions, such as purchase orders and invoices, in a standard format with standard content. As such, it is an important part of an electronic commerce strategy.

When computers exchange data using EDI, the data is transmitted in EDI Standard format so it is recognizable by other systems using the same EDI Standard format. Companies who use EDI must have translator software to convert the data from the EDI Standard format to their computer system's format.

The J.D. Edwards Data Interface for Electronic Data Interchange system acts as an interface between the J.D. Edwards system data and the translator software. In addition to exchanging EDI data, this data interface can also be used for general interoperability and electronic commerce needs where a file-based interface meets the business requirements.

Some benefits of using the Data Interface for Electronic Data Interchange system are:

- Shorter fulfillment cycle
- Increased information integrity through reduced manual data entry
- Reduced manual clerical work

EDI is particularly effective at sending information to multiple applications simultaneously. For more information about the J.D. Edwards EDI solution refer to the *Data Interface for Electronic Data Guide*.

OneWorld APIs

APIs are routines that perform predefined tasks. OneWorld APIs make it easier for third-party applications to interact with OneWorld. These APIs are functions provided to manipulate OneWorld data types, provide common functionality, and provide database access. There are several categories of APIs, including the Common Library APIs and J.D. Edwards Database (JDEBASE) APIs.

Programs using OneWorld APIs are flexible for the following reasons:

- No code modifications are required to upgrade

- When a J.D. Edwards data structure changes, source modifications are minimal to nonexistent
- Common functionality provided through the APIs is less prone to error

When the code in an API changes, typically, business functions simply have to be recompiled and relinked.

COM

COM allows OneWorld to work with third-party applications. This solution is fully compliant with the Microsoft component object model. It is a component based model used to share logic and data. OneWorld also supports DCOM. For more information refer to *COM* in this guide.

CORBA

CORBA allows OneWorld to work with third-party applications. This solution is fully compliant with the Object Management Group's (OMG) component model. It is a component based model used to share logic and data. For more information refer to *CORBA* in this guide.

JAVA

JAVA allows OneWorld to work with third-party applications. It is a component based model used to share logic and data. OneWorld APIs can be exposed for re-use. For more information refer to *JAVA* in this guide.

XML

XML provides a flexible, standards-based protocol for moving data between systems. It allows you to extend enterprise applications and collaborate with business partners and customers. For more information refer to *XML* in this guide.

Table Conversion

Table conversions are a type of batch process that allows you to do high-speed manipulation of data in tables. The table conversion tool allows you to transfer and copy data. You can also delete records from tables.

The table conversion tool can make use of any OneWorld tables, business views and text files, or any tables that are not OneWorld tables but reside in a database supported by OneWorld, such as Oracle, Access, AS/400, or SQL Server. These non-OneWorld tables are commonly referred to as foreign tables.

Fore more information about table conversions refer to the *Table Conversion Guide*.

Messaging Support

J.D. Edwards provides messaging support through business partners. OneWorld can use these messaging systems to handle and pass requests for logic and data. MQ Series is one of the messaging systems OneWorld supports. For more information refer to the *Asynchronous Messaging Programmer's Guide*.

SAP ALE/IDoc

J.D. Edwards supports several third-party and business-partner solutions. This allows information and functional requests to pass between OneWorld and third-party applications.

Open Data Access (ODA)

ODA allows third-party applications direct, read-only access to the OneWorld data tables and active data dictionary. For more information refer to *Open Data Access* in this guide.

API Models

There are several interoperability models available for Using OneWorld APIs. These models are based on a combination of interoperability objectives and processing modes.

The two main objectives to interoperability are:

- Transferring information into OneWorld
- Retrieving information from OneWorld

You can use one of the following processing modes to transfer information into or out of OneWorld:

Synchronous

Synchronous processing implies that you are making a real-time direct call to OneWorld objects. You establish a connection to OneWorld and make direct calls to OneWorld APIs or business functions. The OneWorld object does its work while your calling program waits. Results are immediately available upon completion of the call. Synchronous processing is typically used in interactive applications that require immediate user feedback.

Asynchronous

Asynchronous processing allows an application to submit transactions or requests to another application one at a time. However, during asynchronous processing, the requests are queued and processed without directly connecting to the calling program. An application can submit a request and immediately continue processing without waiting for a result. Results are returned through a separate process when the transaction is complete. Asynchronous processing is typically used when real time feedback is not required, but fairly rapid turnaround on a per transaction basis is still important.

Batch

Batch processing saves transactions over a period of time. A periodic process processes the entire group at once. Batch processing is typically used for large groups of transactions that must be transferred from one system to another daily or monthly.

The available processing modes and the interoperability objectives combine to provide the following interoperability models for OneWorld (inbound refers to transactions into OneWorld and outbound refers to transactions from OneWorld):

- Inbound synchronous

This model uses a master business function that can be called interactively from a third-party application to perform synchronous updates to the J.D. Edwards master or transaction file. The calling program should process any error messages.

- Inbound asynchronous

This model uses an inbound processor batch process that can be automatically activated through the J.D. Edwards subsystem for near real-time processing and to update the J.D. Edwards master or transaction file. Both flat files (ASCII text files) and database files (Z files) in the J.D. Edwards specified format can be supported. The batch process calls the same master business function used by inbound synchronous processing. Error messages are written to the Employee Work Center in the form of action messages. A revision application can be used to correct incoming entries as necessary.

- Inbound batch

This model is identical to inbound asynchronous, except no subsystem processing exists. You must also initiate the inbound processor batch process to update the J.D. Edwards master or transaction file with the external data.

- Outbound synchronous

This model allows you to communicate directly with OneWorld objects in a bidirectional real time mode. You can use a OneWorld business function or API to request information to be returned while you wait. Any return information or feedback is immediately available upon completion of the call.

- Outbound asynchronous

This model uses a master business function that updates the J.D. Edwards master or transaction file and writes a record to a database file (Z file), which can then be processed by an outbound processor batch process automatically activated through the J.D. Edwards subsystem in an asynchronous, or near real-time mode.

- Outbound batch

This model uses an extraction batch process that reads the J.D. Edwards master or transaction file and writes records to a database file (Z file). You use data selection as the only means to specify what records to select for processing.

For more information about using APIs refer to the *API* section in this guide.

This guide includes the following sections:

- Overview
- Connectors
- XML
- APIs
- Appendices

Connectors

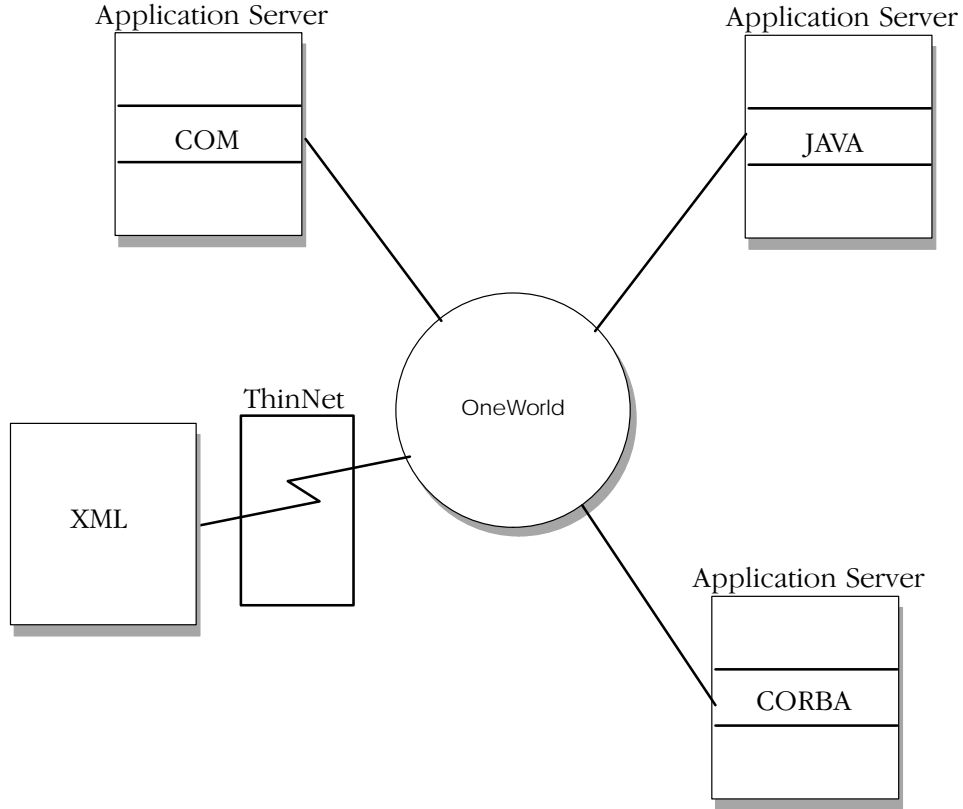
Connectors

This section describes some features that J.D. Edwards makes available for using connectors. J.D. Edwards supports COM, Java, and CORBA connectors. The J.D. Edwards solution also provides session management, point of entry, connection pooling, and transaction functionality.

Although you can use OneWorld APIs directly, using connectors provides additional benefits. Connectors are scalable, multi-threaded, and allow concurrent users.

Choosing a Connector

The following diagram illustrates how different technologies work with OneWorld. It can help you choose which connector you should use, or whether you should use another method, such as XML, instead of a connector. All of these methods allow you to access OneWorld business logic from outside of OneWorld.



The connector you choose depends on which application server you are using.

- If you are using Site Server you should use the COM connector.
- If you are using IBM Websphere, WebLogic, or Bluestone you should use the Java connector.
- If you are using a custom Corba application server, you should use the CORBA connector.

If you can create XML documents on your interoperability server, you can use XML for your interoperability solution. XML provides several benefits:

- XML can be used to aggregate business function calls into one object. This reduces network traffic.
- Because XML uses ThinNet, it is scalable so that multiple connections can be opened.
- XML exposes both APIs and Z files.
- If you create an XML document according to J.D. Edwards specifications, you can transport the document by using ThinNet or an MQ adapter. OneWorld will complete information for the document to provide a response so that you can use messaging.

Generating Business Function Wrappers

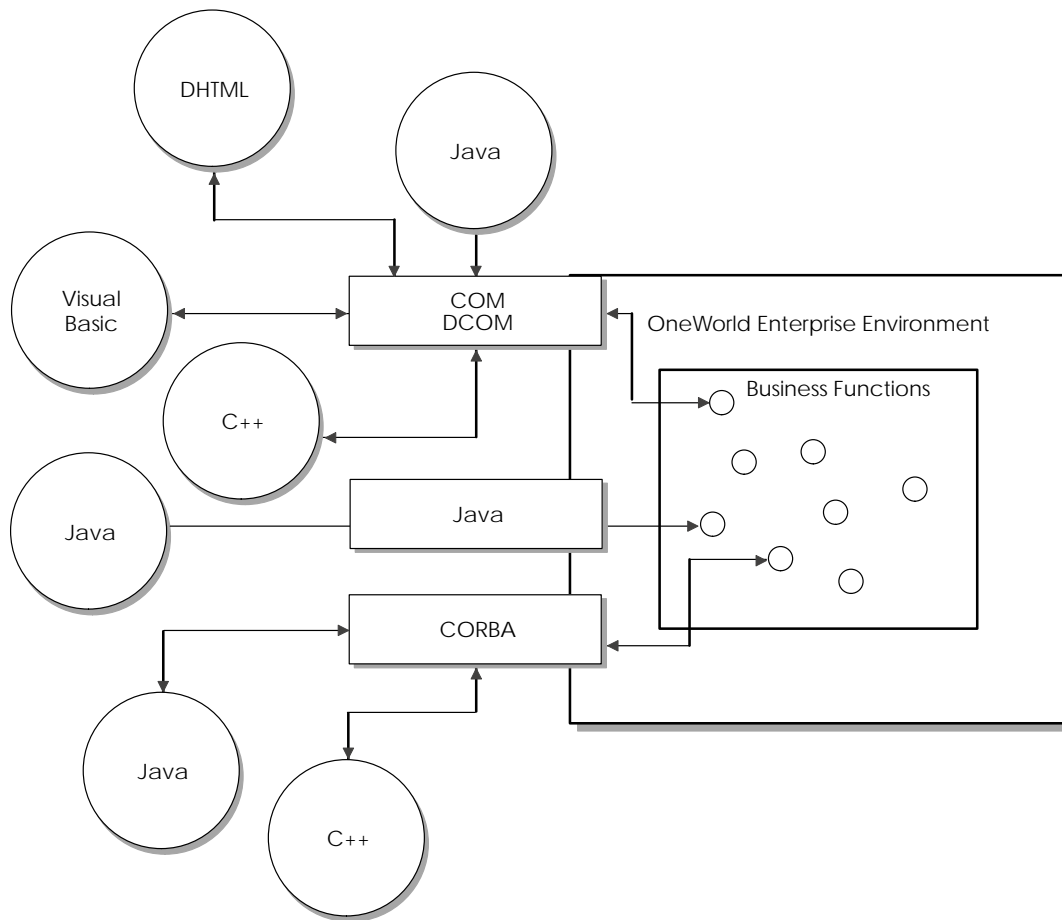
After you decide which connector you want to use you can generate the appropriate business function wrappers. J.D. Edwards provides several generators that you can use to expose OneWorld business logic through common middleware:

- GenCOM
- GenJava
- GenCORBA

These generators are very similar. They are command line tools that you use to wrap OneWorld business functions. GenCOM is only available on NT. You must have a OneWorld client installed on your workstation prior to running GenCOM.

You can use a scripting language, iJDEScript, with the generators.

The following diagram illustrates how the generators work.



The generator uses OneWorld business function specifications to generate the appropriate files to provide middleware wrappers.

COM	*.idl, *.h, *.cpp, *.rc, *.def, Makefile
CORBA	*.idl, *.java
Java	*.java and javadoc HTML files for the wrapper documentation

The following illustration shows a decision flowchart to help you choose which connector you should use.

This section also contains information about the configuration and setup required for COM, CORBA, and Java. It contains the following topics:

- COM
- Java

- CORBA
- iJDEScript
- Interop.ini



COM

This section describes some of the J.D. Edwards features available to help you implement a COM solution. This section also includes setup and configuration. It includes the following topics:

- Understanding the Component Object Model (COM)
- Using the COM Generator (GenCOM)
- Setting up a OneWorld Client Environment
- Installing a COM server on a Non-OneWorld machine
- Setting up a OneWorld DCOM server
- Using the COM Wrapper Version Checker (CheckVer)
- COM tracing and logging



Understanding the Component Object Model (COM)

The Component Object Model (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 primarily 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 programming language independent. 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.

DCOM enables COM objects in a distributed environment.

OneWorld and COM

Using COM, OneWorld exposes all master and major business functions through the Interface Definition Language (IDL) standard. With COM, OneWorld 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.

OneWorld COM Objects

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

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

OneWorld COM Interoperability Usage

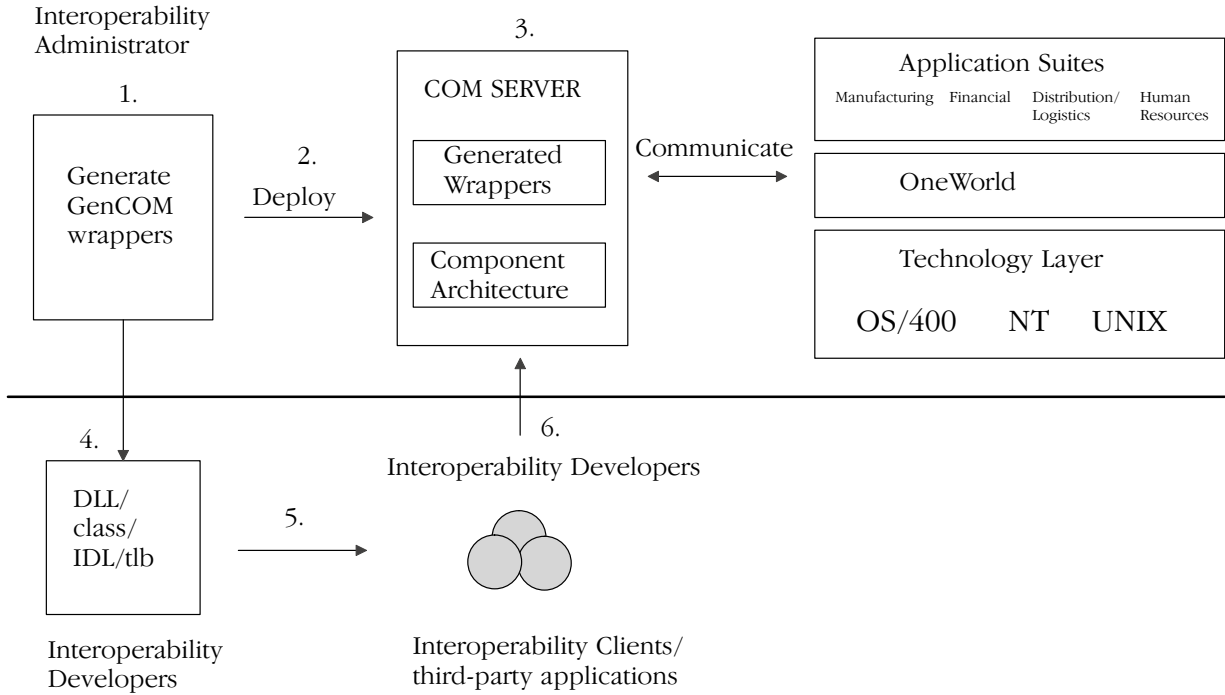
The following steps illustrate how the OneWorld COM interoperability solution typically flows.

1. The administrator generates the COM wrappers.
2. The administrator deploys the COM objects to the COM server.

The COM server allows communication with the application server so that the generated COM objects can be used in applications.

3. Once the COM objects are on the COM server, the COM objects are configured to communicate with the application server.
4. The DLLs or IDLs from the generated COM objects are copied so that developers can use them.
5. Application developers create the applications.
6. The applications can then communicate with the COM server.

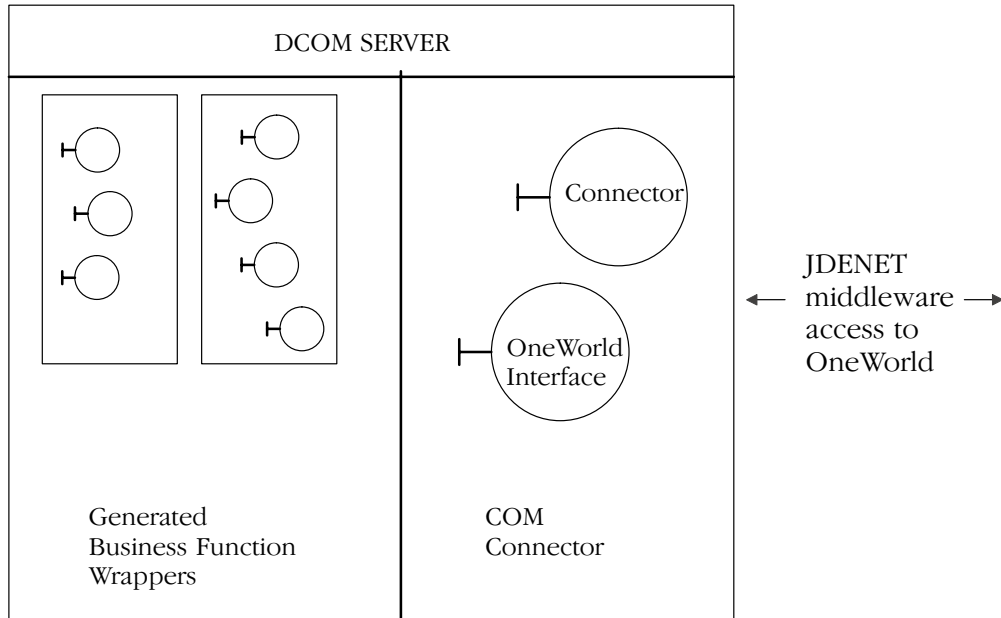
The following diagram illustrates how the OneWorld COM interoperability solution flows.



OneWorld COM Server

The OneWorld COM server contains two parts:

- COM Connector
- Generated OneWorld COM components (wrappers)



COM Connector

The OneWorld COM server provides an interface to OneWorld, executes business functions within valid transactions, and provides error processing for interoperability clients. The main component of the OneWorld COM server is the COM Connector. The COM Connector provides COM components that interface with OneWorld 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 OneWorld. It manages all user sessions connected to the COM server. The following binaries combine to comprise the COM Connector:

JDECOMConnector2.exe

JDECOMMN.dll

Callobject.dll

Comlog.dll

Xmlinterop.dll

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

The COM Connector is available with the OneWorld enterprise server CD.

Generated COM Components

GenCOM is included in the OneWorld 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 OneWorld business functions. Once the generated wrapper components are registered to the COM environment they can be used to access OneWorld business function functionality.

Refer to *Understanding iJDEScript* for more information about iJDEScript.

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.

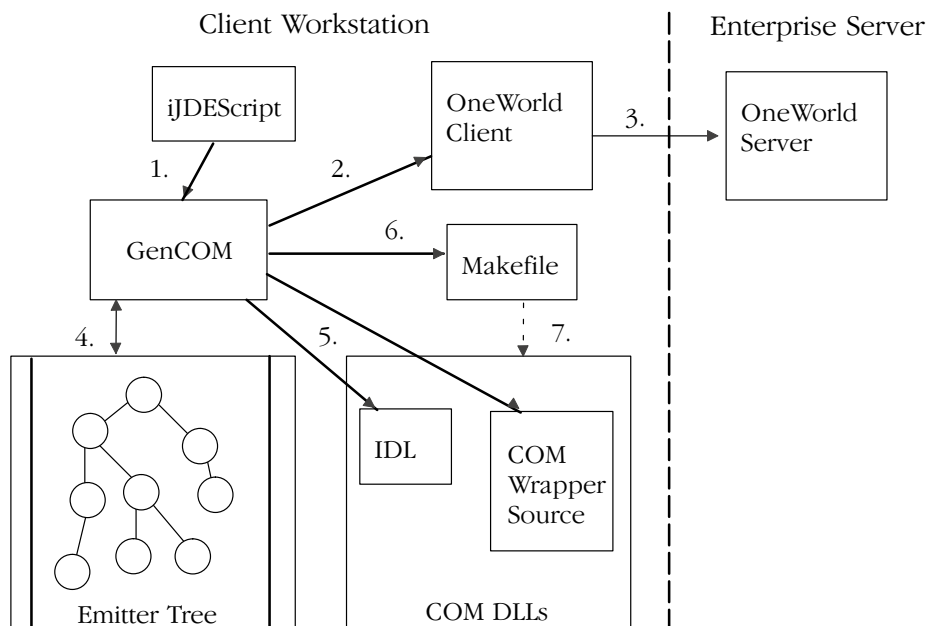
You should take additional precautions to provide further reliability. After you use the COM connector to enter an order or execute a business function, your process should:

- 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.

OneWorld GenCOM

GenCOM is a OneWorld client tool that generates OneWorld COM components. It 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.

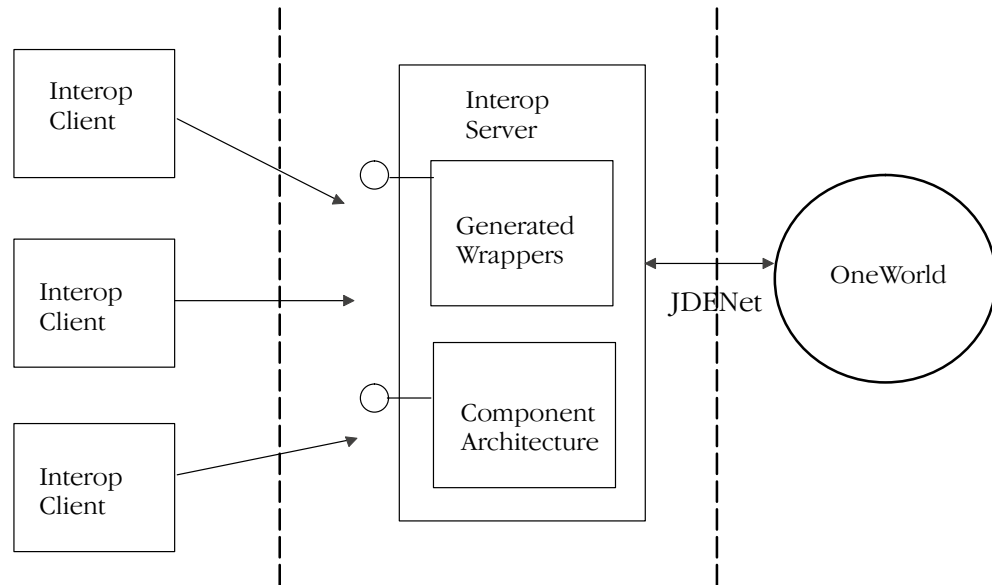
1. GenCOM reads the iJDEScript file.
2. GenCOM fetches the metadata for the business functions specified in the iJDEScript file.
3. GenCOM resolves dependency on the data structure.
4. GenCOM creates an internal emitter tree for the library to be generated.
5. GenCOM reads each node of the internal emitter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



Refer to *Using the COM Generator (GenCOM)* for more information about GenCOM.

OneWorld COM Server Deployment

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



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 OneWorld 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. There are several options available for generation. The COM Generation Tool is located in <install>\system\bin32\GenCOM.exe.

Syntax

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

Example

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

```

[e:\proj\b733_sp6\system]cd bin32
[e:\proj\b733_sp6\system\bin32]gencom /?
Usage: GenCOM [options] [libraries]
options:
/?                Display this help information
/C++ <option>    Pass <option> to the C++ compiler
/Cat <category>  Generate only <category> function wrappers
/CL <file>       Use <file> as the C++ compiler
/Cmd <file>      Process commands from <file>
/Cmd *           Process commands from the console
/D name value    Define a macro value
/EnvironmentID <env> Use <env> to log into OneWorld
/ErrFile <file>  Use <file> to record all error messages
/MIDL <option>   Pass <option> to the MIDL compiler
/MIL <file>      Use <file> as the MIDL compiler
/Listlibraries   List the libraries in f9860
/MsgFile <file> Use <file> to record all messages
/NoSIFs         Generate wrappers for parametersets only
/Out <path>      Use <path> for all output files
/Password <password> Use <password> to log into OneWorld
/ProgID <format> Use <format> for ProgID's
/TempOut <path>  Use <path> for all temporary files
/UserID <userid> Use <userid> to log into OneWorld
/UIProgID <format> Use <format> for version indep. ProgID's
/SERVER <Intel|Alpha> Generate DLL that will be deployed in OneWorld Server Environment
environment
category:
1 - Master Business Functions
2 - Major Business Functions
3 - Minor Business Functions
- - Uncategorized Business Functions
format:
%%macro%...      Macros (%macro%) are replaced by value
predefined macros:
%name%           The name of the current entity
%release%        The current release of OneWorld
%version%        The current version of OneWorld
%environment%    The current OneWorld environment
Example:
GenCOM /ProgID %name%.1 /Cat 1 /Cat 2 CAEC
generates wrappers for all category 1 and 2 business functions in
the CAEC library
[e:\proj\b733_sp6\system\bin32]

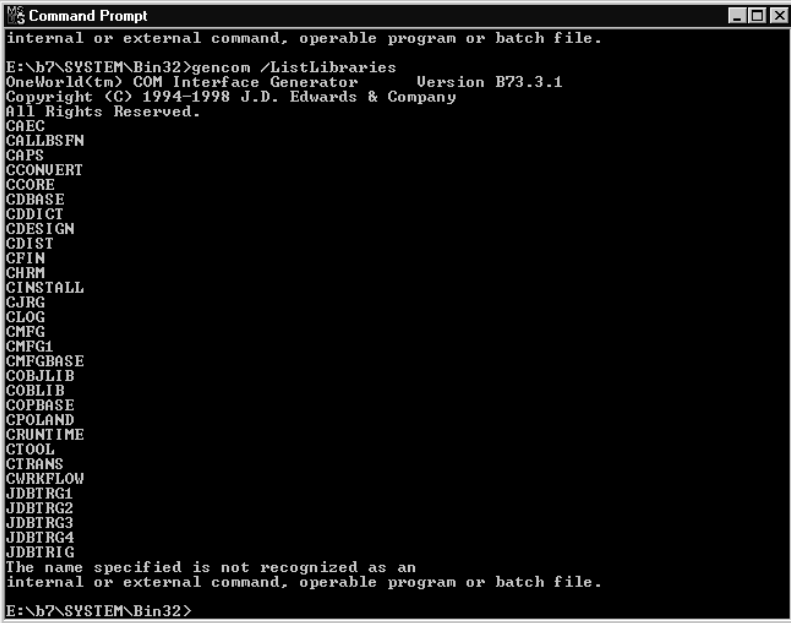
```

Options

- `/?` Lists the options available for generation.
- `/C++ <option>` Provides GenCOM with the compiler options you wish 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. Refer to *Understanding iJDEScript* for more information.
- `/Cmd <filename>` Processes code generation commands from <filename>. Refer to *Understanding iJDEScript* for more information.

<code>/Debug</code>	Builds debug information (.pdb and .bsc files) into the libraries so that the Visual Studio debugger can access source information.
<code>/EnvironmentID <env></code>	Provides GenCOM with the environment in which you wish to log into OneWorld.
<code>/ErrFile <file></code>	Provides GenCOM with the filename to log errors produced by GenCOM during the generation process, for example, "errors.log".
<code>/MIDL</code>	Provides GenCOM with the MIDL compiler options you wish to use in the generation of the COM servers.
<code>/MTL <file></code>	Tells GenCOM which MIDL compiler (.exe) to use for compilation.
<code>/ListLibraries</code>	Lists all the available libraries that you can run GenCOM against.
<code>/MsgFile <file></code>	Provides GenCOM with the filename to log messages produced by GenCOM during the generation process, for example, "messages.log"
<code>/NoBSFN</code>	Tells GenCOM not to create wrappers for business functions. This option is for generating parameter sets only.
<code>/NoCompile</code>	Tells GenCOM to only generate the source files without compiling.
<code>/NoDebug</code>	Optimizes libraries for space using the /O1 Visual C++ compiler option.
<code>/Out <path></code>	Provides GenCOM with the directory (path) in which to place the output files, for example "C:\winnt\system32".

<code>/OWRelease</code> 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.
<code>/Password</code> <password>	Provides GenCOM with the password with which you wish to log into OneWorld.
<code>/STA</code>	Generates STA components. (By default, all generated components are MTA and are optimized for scalability and performance. <code>/STA</code> allows you to generate STA components if you need them.)
<code>/TempOut</code> <path>	Provides GenCOM with the directory (path) in which to place temporary files needed for the build process, for example, "C:\temp."
<code>/UserID</code> <userid>	Provides GenCOM with the username with which you wish to log into OneWorld.



```
MS Command Prompt
internal or external command, operable program or batch file.
E:\b7\SYSTEM\Bin32>gencom /ListLibraries
OneWorld(tm) COM Interface Generator      Version B73.3.1
Copyright (C) 1994-1998 J.D. Edwards & Company
All Rights Reserved.
CAEC
CALLBSFN
CAPS
CCONUERT
CCORE
CDBASE
CDDICT
CDESIGN
CDIST
CFIN
CHR
CINSTALL
CJRG
CLOG
CMFG
CMFG1
CMFGBASE
COBJLIB
COBLIB
COPBASE
CPOLAND
CRUNTIME
CTOOL
CTRANS
CWRMFLW
JDBTRG1
JDBTRG2
JDBTRG3
JDBTRG4
JDBTRIG
The name specified is not recognized as an
internal or external command, operable program or batch file.
E:\b7\SYSTEM\Bin32>
```


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 OneWorld program-independent, and is based on the library and the interface specifications in the script file. The key OneWorld release contains the OneWorld release and environment information. For example, if the library name is SalesOrderEntry and the interface name is JDESalesOrderEntry, then the progID will be SalesOrderEntry.JDE SalesOrderEntry. If GenCOM is run on B733.SP7_COM and the environment is CRPB733, then the OneWorldRelease key will contain B733.SP7_COM.CRPB733. If there is a type mismatch you will receive a warning.

The CompatibleEnvironment key remembers the list of OneWorld environments that the wrapper is compatible with. 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\{5B5D5884-7B76-11D3-A722-00105A1C00F4}
  \InprocServer32   D:\b7\system\dcom\sodll\SalesOrderEntry.dll
  \ProgID           SalesOrderEntry.JDESalesOrderEntry
  \VersionIndependentProgID  SalesOrderEntry.JDESalesOrderEntry
  \OneWorldRelease  B733.SPJ.ADEVHPO2
  \CompatibleEnvironment  PRODB733      1
```

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

You must have a fully licensed copy of OneWorld properly installed on the target machine. You must also log on to the OneWorld 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 simply 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
Dim ParentAddress As D0100381 'Data Source
Dim I As Long 'server Session ID
Private Sub Form_Load()
I=conn.Login("Foo","Bar","CRPB733")
Set OW = conn.CreateBusinessObject("OneWorld.FunctionHelper.I",1)
Set AB = conn.CreateBusinessObject("AddressBook.JDEAddressBook",1)
Set phone = AB.CreateGetPhoneParameterset
Set Mailing = AB.CreateGetMailingNameParameterset
Set Address = 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"
#include "AddressBook_i.c"
#include "jdecomconnector2.h"
#include "jdecomconnector2_i.c"
#define IPhone ID0100032
#define IMailing ID0100031
#define IAddress ID0100033
#define IEffectiveDate ID0100019
#define IParentAddress ID0100381
#define SERVER OLESTR("COMSRV") //Change to the COM server.
#define ABNO 4242 // change this according to user input.
HRESULT CreateConnector( IConnector **ppConnector )
{
    HRESULT hr = E_FAIL;

    *ppConnector = 0;

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

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

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

    printf("Login started\n");
    _bstr_t User(L"Foo "), Password(L"Bar "), Env("CRPB733");

```

```

    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 ) || ( !low ) )
    {
        printf( "CreateBusinessObject( OneWorld.FunctionHelper.1 ) failed with
hr %x", hr );
        return E_FAIL;
    }

    hr = idsptch->QueryInterface(IID_IOneWorldInterface, ( void **)ow );
    if ( ! SUCCEEDED( hr ) || ( !low ) )
    {
        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 *ac-
cessno )
{
    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 ) || ( !ab ) )
    {
        printf(" "CreateBusinessObject( AddressBook ) has failed with hr %x", hr
);
        return E_FAIL;
    }
    return S_OK;
}

```

```

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 );
            UseAddressBook( pConnector, ow, &accessno );
        }
        //Do more processing with AddressBook and logoff at the end.
    }
    CoUninitialize();
}

```

The code above creates the connector object and uses it to create a J.D. Edwards business function and its associated ParameterSet. It 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.

Using BHVRCOM via COM

OneWorld 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
... ..
```

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 OneWorld system. You should not assume that a client has installed these servers as part of the standard OneWorld installation. You should provide a full installation of any of the servers your applications require.

J.D. Edwards also provides a OneWorld COM generation tutorial. The tutorial is a step-by-step study on how to build an application and goes through the install issues for a Visual Basic application. If you are developing with a different software package, please consult the documentation for that product.

Setting Up a OneWorld Environment for GenCOM

There are several steps you must take to set up an NT client environment. You should make sure the following are set up appropriately:

- Include directories
- Lib directories
- MSDev directories
- Paths

Include Directories

< 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\mfcc\include

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

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

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

< Directory where JDEdwards OneWorld is located and release either Master, Prod, or Pristine >\include

Example 1: D:\B7\MSTB733\include

Example 2: D:\B7\PROD\include

< Directory where JDEdwards OneWorld is located and release either Master, Prod, or Pristine >\includeV

Example: D:\B7\SYSTEM\includeV

< Directory where JDEdwards OneWorld is located and release either Master, Prod, or Pristine >\include

Example: D:\B7\SYSTEM\include

Lib directories

< 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 JDEdwards OneWorld is located > \System\Lib32

Example: D:\B7\System\Lib32

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

Paths

< 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 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 JDEdwards OneWorld is located>\System\Bin32

Example: D:\B7\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 OneWorld server and the OneWorld client are different.

Installing a COM Server on a Non-OneWorld Machine

There are several steps you must follow to install a COM server on a non-OneWorld machine.

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\JDEdwards on a non-OneWorld machine.

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

2. Execute the following command on the target location to register the COM connector components.

```
c:\program files\JDEdwards\JDECOMConnector2.exe /RegServer
```

3. Run GenCOM on a OneWorld client machine and copy the output DLL and the wrapper components (for example, wrapper.dll) to this directory.

```
c:\program files\JDEdwards\wrapper.dll
```

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

```
c:\program files\JDEdwards\regsvr32 wrapper.dll
```

5. Create the JDEinterop.ini file.

Set the enterpriseServer and port values to the OneWorld application server with which you want the COM server to communicate. Refer to *Understanding JDEinterop.ini Settings* for more information about the JDEinterop.ini file.

Your DCOM server is now ready on this machine.

Setting Up a OneWorld DCOM Server

There are several steps you must take to set up a DCOM server on a OneWorld enterprise server machine. To ensure that the interoperability client works properly with the OneWorld DCOM server, you must perform several steps, including:

- Set up DCOM for a server environment
- Set up DCOM for a client environment

Setting Up DCOM for a Server Environment

1. Run GenCOM on a OneWorld client machine, with the following options:
“gencom /out <path> /tempout <path> /cmd App.cmd”

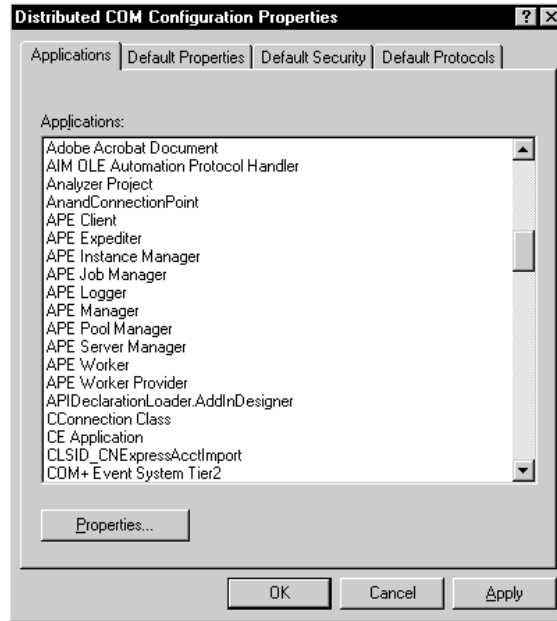
GenCOM is a OneWorld client side only tool. That is why you must perform this step on a OneWorld client machine.

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 “jdecocomconnector2.exe /RegServer”
 - Run “regsvr32 *App.dll*”
 - Set the correct security level for *jdecocomconnector2.exe* and *App.dll*

Setting Up Security on the COM Server

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

The following form appears.

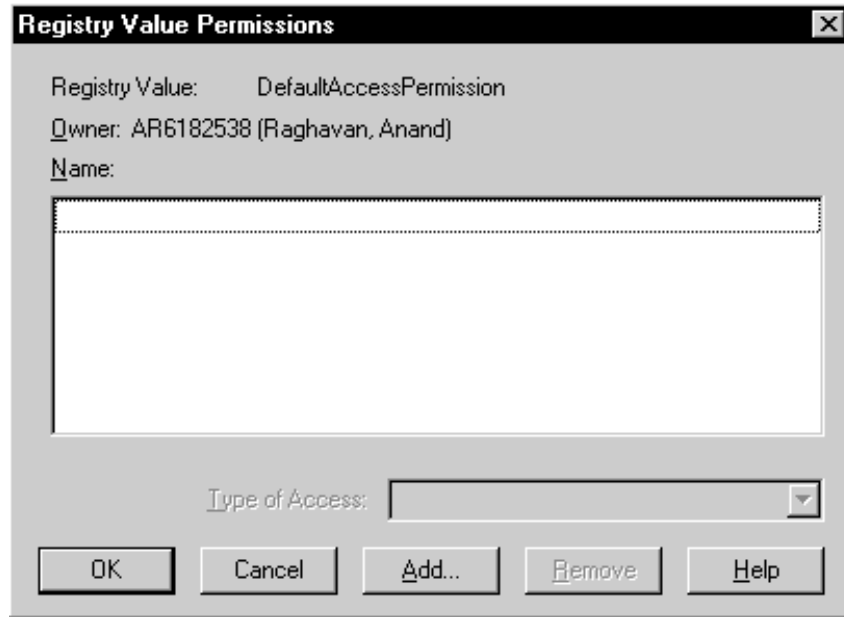


3. Click the Default Security tab.

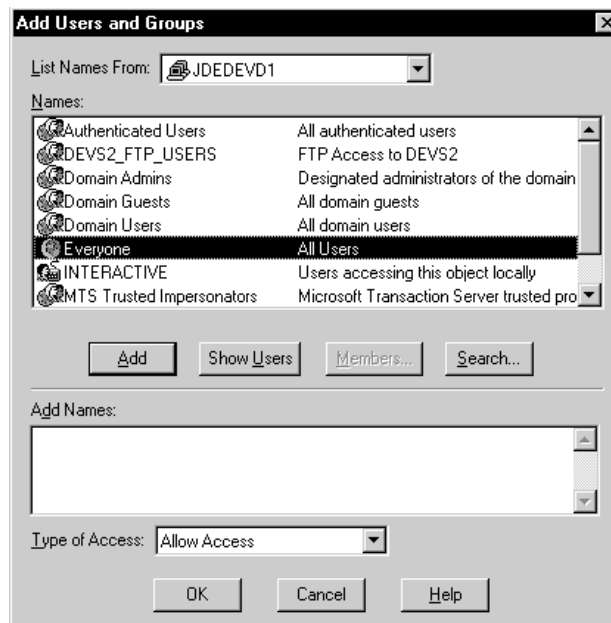


4. Click on the Edit Default Button in Default Access Permissions group.

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.

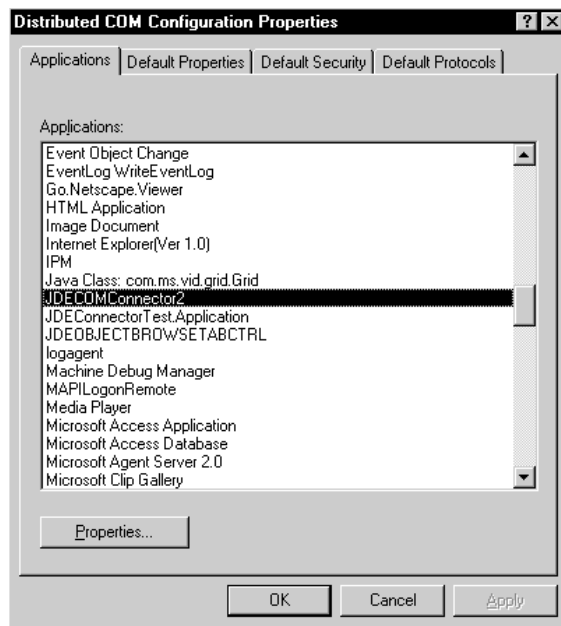
7. Click OK.

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

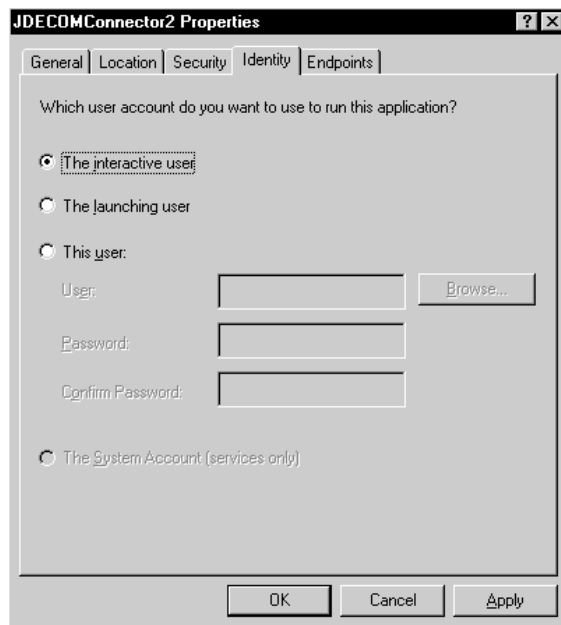
Set up the identity as Interactive user.

► 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.

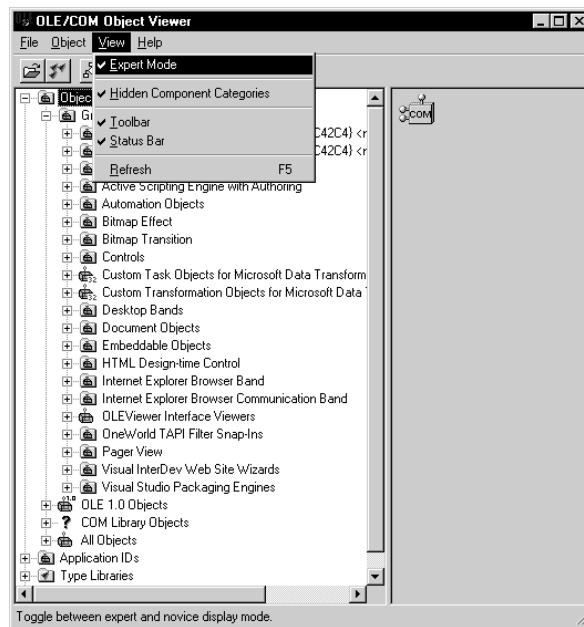
4. Click Apply to apply the change.

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.

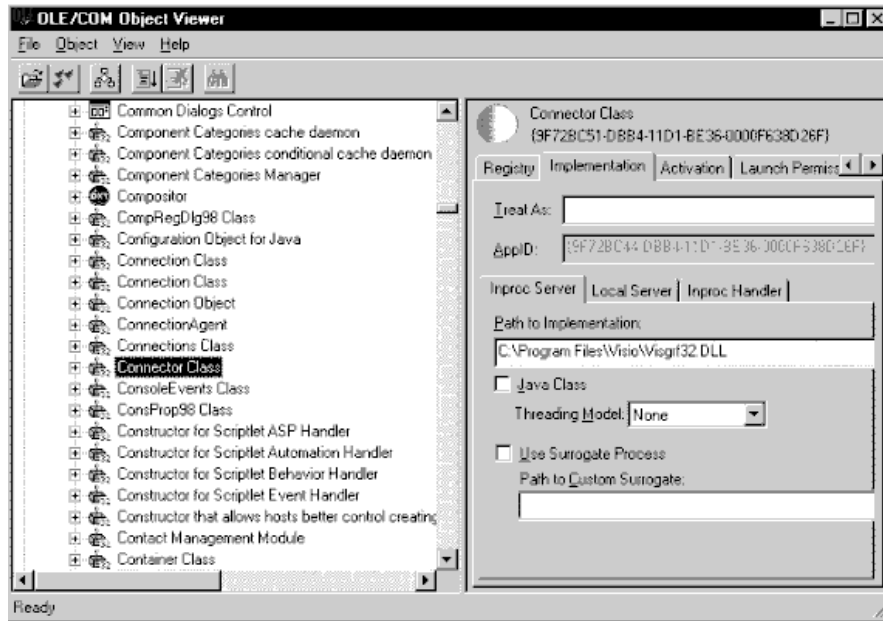
To use Callbacks (Connection Points) with the COM solution, repeat the same procedure on the COM client machine. Most of the shipped examples use Callbacks and require that you open the security on the client machine.

Setting Up DCOM for a Client Environment

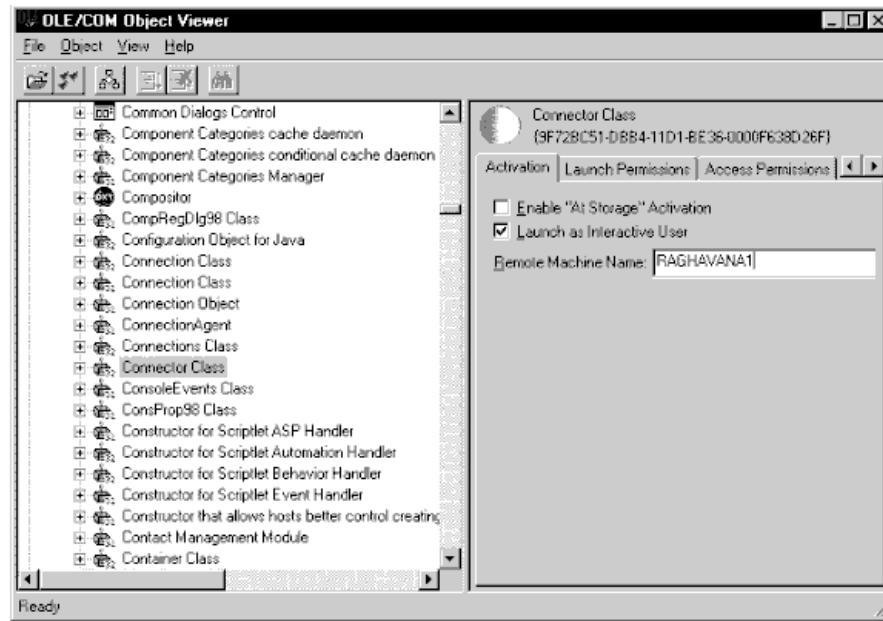
1. On the DCOM client machine run “jdecomconnector2.exe /RegServer”.
2. From the menu bar, choose oleview.
3. Click View and choose Expert Mode.
4. In the oleview window under Object Classes, double click All Objects, and wait for all objects to appear



5. Under All Objects, find Connector Class and click it.
6. Click the Implementation tab on the right side panel, and then click the local server and remove anything that appears in the editing window.



7. On the Activation tab, turn on the Launch as Interactive User option.
8. In Remote Machine Name, enter the COM server machine name.



9. Repeat steps 5–8 for MathNumeric Class.
10. Start the DCOM client application.

Remember client-only OneWorld 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 OneWorld system administrator performs this task.

The xml files generated by GenCOM are the signatures of the objects generated against specific OneWorld 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 OneWorld 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 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 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

You run CheckVer from the command line on the COM server. There are several options available for generation.

Syntax

```
CheckVer [options] filename
```

Example

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

Options

-r	CheckVer will only report whether the environment is compatible with the server but won't update the COM server with the result.
-----------	--

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 OneWorld logging format. It includes the Time Thread ID [User ID] and the Description, for example:

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

Refer to *Understanding jdeinterop.ini Settings* for more specific information about setting in the `jdeinterop.ini` file.

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 on the Connector System Tray Icon.
- Select the menu item `ChangeIniSettings`.

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

Troubleshooting

Tracing impacts performance. You do not need tracing on unless you are debugging. If you experience a high performance impact, 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.



Java

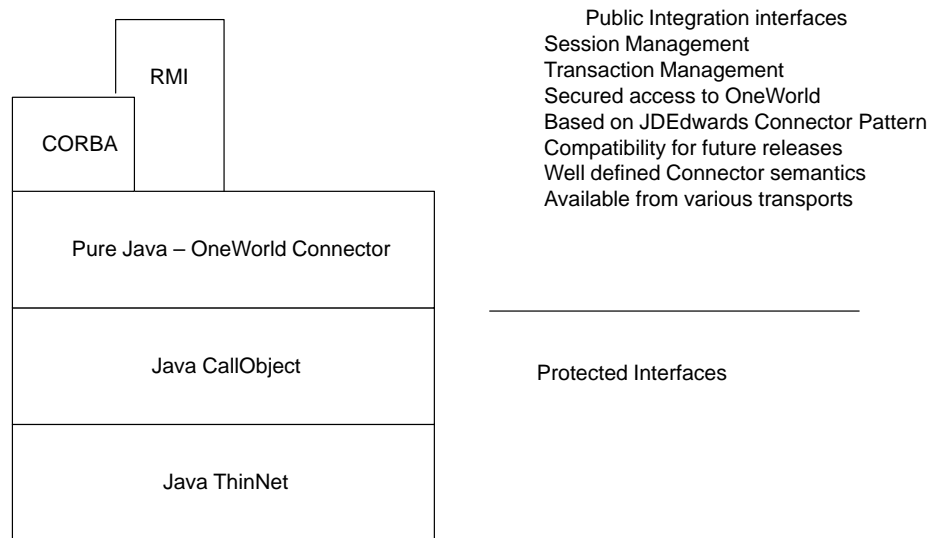
This section describes some of the J.D. Edwards features available to help you implement a Java solution. This section also includes setup and configuration. It includes the following topics:

- Understanding Java
- Using the Java Generator (GenJava)
- Setting up a OneWorld Client Environment for GenJava
- Installing Java components on a Non-OneWorld Machine
- Understanding Java and CORBA versioning
- Using the Java Wrapper Version Checker (CheckVer)



Understanding Java

Pure Java interoperability allows you to write Java applications to interact with OneWorld. You can also put CORBA, Java RMI on top of the Pure Java layer as illustrated by the following diagram.



Java and OneWorld

A OneWorld business function is a logical collection of C functions and their associated data structures grouped together to produce a unit of work. OneWorld 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 OneWorld business functions. Because all methods must be defined in a Java class, a library must be defined in the corresponding *iJDEScript* file. Refer to *Understanding iJDEScript* for more information about *iJDEScript*.

For example, if within a OneWorld library A, there exists a business function “B550001” and within this business function there exist two C functions named “foo1” and “foo2” with data structures for each function named “DS1” and “DS2”, the corresponding OneWorld Java class would be:

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

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

Each data structure in OneWorld 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 Mathnumeric in addition to the primitive data types. The two Java classes JDEDate and JDEMathnumeric are defined respectively. All public APIs can be also found in the Java document ConnectorDoc.jar shipped with the product.

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 there should be a class for the data structure with the public method getmnAddressBook that returns a JDEMathNumeric object. Then you use DS.getmnAddressBook().setValue("1") to set the mnAddressBook value to 1 in the data structure.

Understanding GenJava

J.D. Edwards provides a Java Generation tool, GenJava that you can run to expose OneWorld business functions through Java. The OneWorld 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 B7\system\classes directory or any directory redirected by GenJava.

Pure Java Component Usage

You must deploy the pure Java framework components, business function wrappers, configuration files, and jdeinterop.ini file to the location where you want to use them. The framework components include two JAR files, Connector.jar and Kernel.jar. ConnectorDoc.jar is a compressed file containing the Java documents for all the Connector classes. To determine which public API can be used in the Connector package, unjar or unzip the jar file.

You must also have the appropriate settings in the jdeinterop.ini file. This file allows the pure Java framework components to interact with the OneWorld enterprise server. Refer to *Understanding jdeinterop.ini Settings* for information about these settings.

Kernel.jar and Connector.jar must be added to the CLASSPATH. To run a Java application, you need to set the system property config_file to point to the right location of jdeinterop.ini, such as,

```
Java -Dconfig_file=d:\system\classes\jdeinterop.ini abApplication
```

Refer to *Using the Java Generator (GenJava)* on how to write a Java application.

Using the Java Generator (GenJava)

The OneWorld Java generation tool, GenJava, provides access to OneWorld business functions by generating pure Java interfaces for OneWorld 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]
```

Example

```
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 OneWorld, including the UserID, Password, and environment.

```

C:\WINNT\System32\cmd.exe
D:\work\Java Example\Addressbook>genJAVA /?
J.D. Edwards OneWorld(tm) Interoperability Interface Generator
Copyright (C) 1996-2000 J.D. Edwards World Source Company
All Rights Reserved.
Usage: genJAVA [options] [libraries]
options:
/?          Display this help information
/Cat <category>  Generate only <category> function wrappers
                Supported categories:
                /'1/' - Master Business Functions
                /'2/' - Major Business Functions
                /'3/' - Minor Business Functions
                /'-/' - Uncategorized Business Functions

/Cmd <file>     Process commands from <file>
/Cmd *         Process commands from the console
/Compiler <file> Use <file> to compile java files
/D name value  Define a macro value
/EnvironmentID <env> Use <env> to log into OneWorld
/ErrFile <file> Use <file> to record all error messages
/ListLibraries List the libraries in Object Librarian Table
/MsgFile <file> Use <file> to record all messages
/NoBSFN       Generate wrappers for parametersets only
/Out <path>   Use <path> for all output files
/Password <password> Use <password> to log into OneWorld
/TempOut <path> Use <path> for all temporary files
/UserID <userid> Use <userid> to log into OneWorld

Example:
genJAVA /Cat 1 /Cat 2 CAEC
- Generates JAVA wrappers for all category 1 and 2 business functions in
  the CAEC library

```

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. Refer to <i>Understanding iJDEScript</i> for more information.
/Cmd <filename>	Processes code generation commands from <filename>. Refer to <i>Understanding iJDEScript</i> for more information.
/Compiler <file>	Uses <file> to compile Java files.
/D name value	Defines a macro value
/EnvironmentID <env>	Uses <env> to log into OneWorld
/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 wish to log into OneWorld.
/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 OneWorld.
/XMLOnly	Generate only the XML file.

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

```

MS-DOS Prompt
C:\>genJava /ListLibraries
J.D. Edwards OneWorld(tm) CORBA Interface Generator
Copyright (C) 1996-2000 J.D. Edwards World Source Company
All Rights Reserved.
CAEC
CALBSPN
CBUSPART
CCONVERT
CCORE
CCRIN
CDBASE
CDDICT
CDESIGN
CDIST
CFIN
CHR
CINSTALL
CINU
CLOC
CLOG
CMFG
CMFG1
CMFGBASE
COBLIB
COBLIB
COBASE
CRES
CRUNTIME
CSALES
CTOOL
CTRAN
CTRANS
CWARE
CURRELOW
JDBTRG1
JDBTRG2
JDBTRG3
JDBTRG4
JDBTRIG

```

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

```
GenJava /cmd AddressBook.cmd
```

This prompts a OneWorld sign-on window for you to enter the user ID, password, 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.

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 B7/System/classes at the drive of the OneWorld 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 OneWorld 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 OneWorld using a valid OneWorld user ID, password and environment name. The environment must be valid on the OneWorld enterprise server.
3. Get the OneWorldInterface object reference by calling CreateBusinessObject() with an object name, such as Connector::OneWorldInterface.
4. Get the object reference for the wrapper for the OneWorld business function generated by GenJava, for example AddressBook. The object

name passed into 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

Error = 1

Warning = 2

Process the data returned by the business function.

8. Log off OneWorld.

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;
        int l = 0;
1.         connectorProxy = new Connector();
2.         try {
            l = connectorProxy.Login("FOO", "BAR", "PDEVHPO2");
            System.out.println("Log in successfully");
        } catch (reject r) {
            System.out.println("got reject exception");
            String s = r.reason;
            System.out.println(s);
            System.exit(1);
        } catch (Exception e) {
            System.out.println("got other exception");
            e.printStackTrace();
            System.exit(1);
        }
        try {
3.         ow = (OneWorldInterface) connectorProxy.CreateBusinessOb-
ject("Connector::OneWorldInterface", l);
            System.out.println("got OneWorldInterface");
        } catch (reject r) {
            String s = r.reason;
            System.out.println(s);
            return;
        }
        //create AddressBook object
        try {
4.         ab = (AddressBook)connectorProxy.CreateBusinessOb-
ject("JDEAddressBook::AddressBook", l);
            System.out.println("got AddressBook");
        } catch (reject r) {
            String s = r.reason;
            System.out.println(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, l);
        } catch (reject e) { System.out.println(e.reason); }
    }
}

```

```
if (i != 1) {  
    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 == 2) {  
        System.out.println("warning count is" + ow.GetWarningCount());  
        for ( int j = 0; j < ow.GetWarningCount(); j++) {  
            String s = ow.GetWarningAt(j);  
            System.out.println("warning" + j + ":" + s);  
        }  
    }  
    } else {  
        for (int j = 0; j < ow.GetErrorCount(); j++) {  
            String s = ow.GetErrorAt(j);  
            System.out.println("error" + j + ":" + s);  
        }  
        System.out.println(" BSN error");  
    }  
  
    // log off  
8. connectorProxy.Logoff(l);
```


Setting Up a OneWorld Client Environment for GenJAVA

There are several steps you must take to set up a OneWorld 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 JDEdwards OneWorld is located>\System\classes\Kernel.jar

<Directory where JDEdwards OneWorld is located>\System\classes\Connector.jar

Installing JAVA Components on a Non-OneWorld Machine

There are several steps you must follow to install JAVA components so that you can run a JAVA application on a non-OneWorld 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:\JDEdwards\Interop on a non-OneWorld machine.
 - Kernel.jar
 - Connector.jar
 - Jdeinterop.ini
2. Create a separate repository directory for business object jar files.
3. Run GenJAVA on the OneWorld client machine and copy the output jar file (for example, JDEAddressBook.jar) to this directory.
4. Add Kernel.jar and Connector.jar to the CLASSPATH.
5. Depending on whether you want the library in static mode or dynamic mode, put the JAR file in the CLASSPATH.
6. Edit Jdeinterop.ini for proper settings.
7. Run the JAVA application with the system property config_file pointing to the location of jdeinterop.ini. For example, java -Dconfig_file=c:\JDEdwards\Interop\jdeinterop.ini AddressBookApplication.

Understanding JAVA and CORBA Versioning

Business object wrappers that are generated against one environment may not be compatible with another environment. Versioning prevents you from creating Java and CORBA business objects unless the environment used at login is the same as used 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. There is no 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.

Static and Dynamic Modes

The CORBA interoperability server should always be run in dynamic mode.

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 on the fly. 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. There is no limit to the number of simultaneous business object library versions. The old library definitions remain in the virtual machine until there are no more references to the old business objects. This can significantly affect memory usage in the virtual machine.

Using the JAVA Wrapper Version Checker (CheckVer)

You can run CheckVer to verify whether a previously generated Java or CORBA business object library is compatible with another environment. Typically the OneWorld system administrator performs this task. The xml files generated by GenJava and GenCORBA are the signatures of the objects generated against specific OneWorld 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 OneWorld environment, you run GenJava or GenCORBA 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 or CORBA server 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, and xml.jar files are in the CLASSPATH. This can be done either with the CLASSPATH environment variable, or on the command line itself. The examples below assume the CLASSPATH environment variable is set.

Syntax

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

Example

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

Running CheckVer (GenCORBA)

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, JDECORBAConnectorOrbixWeb.jar, OrbixWeb.jar, and xml.jar are in the classpath. This can be done either with the CLASSPATH environment variable, or on the command line itself. The examples below assume the CLASSPATH environment variable was set.

Syntax

```
Java
-Dcom.jdedwards.system.corba.ORBWrapperClass=com.jdedwards.system.corba.
OrbixORBWrapper
-Dorg.omg.CORBA.ORBClass=IE.Iona.OrbixWeb.CORBA.ORB
-Dorg.omg.CORBA.ORBSingletonClass=IE.Iona.OrbixWeb.CORBA.singletonORB]
com.jdedwards.system.connector.CheckVer [jarfile] [xmlfile]
```

Example

```
Java
-Dcom.jdedwards.system.corba.ORBWrapperClass=com.jdedwards.system.corba.
OrbixORBWrapper
-Dorg.omg.CORBA.ORBClass=IE.Iona.OrbixWeb.CORBA.ORB
-Dorg.omg.CORBA.ORBSingletonClass=IE.Iona.OrbixWeb.CORBA.singletonORB]
com.jdedwards.system.connector.CheckVer JDEAddressBookInterop.jar
JDEAddressBook.xml
```



CORBA

This section describes some of the J.D. Edwards features available to help you implement a CORBA solution. This section also includes setup and configuration. It includes the following topics:

- Understanding CORBA
- Using the CORBA Generator (GenCORBA)
- Setting Up a OneWorld Client Environment for GenCORBA
- Setting Up an Environment for CORBA



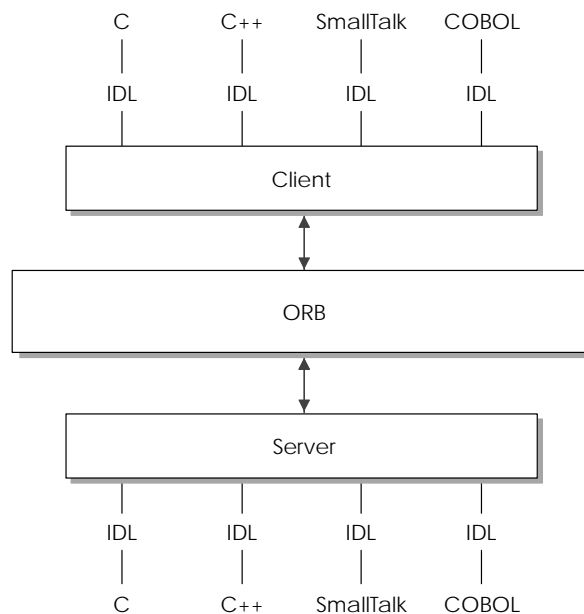
Understanding CORBA

The Common Object Request Broker Architecture (CORBA) allows you to use objects independent of language or platform. CORBA is a standard defined by the Object Management Group (OMG), it is not a product. There are several vendors that use CORBA standards to provide Object Request Brokers (ORBs) that implement these standards.

CORBA is programming language independent. Unlike software libraries or DLLs that are compiled to specific language or linkage conventions, CORBA-based software components are created ready to work with any CORBA client. For example, a Visual C++ or Java application can use CORBA objects created in Visual Basic or Java.

CORBA provides the following benefits:

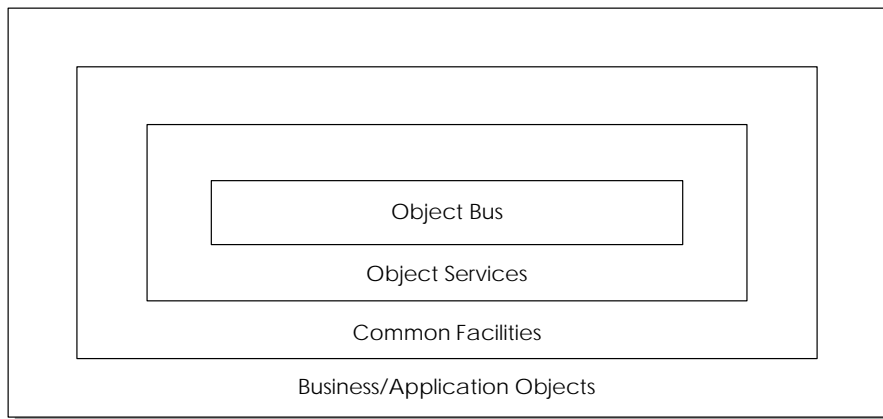
- Object-oriented approach for reuse
- Integration
- Standards
- Portability
- Manageability



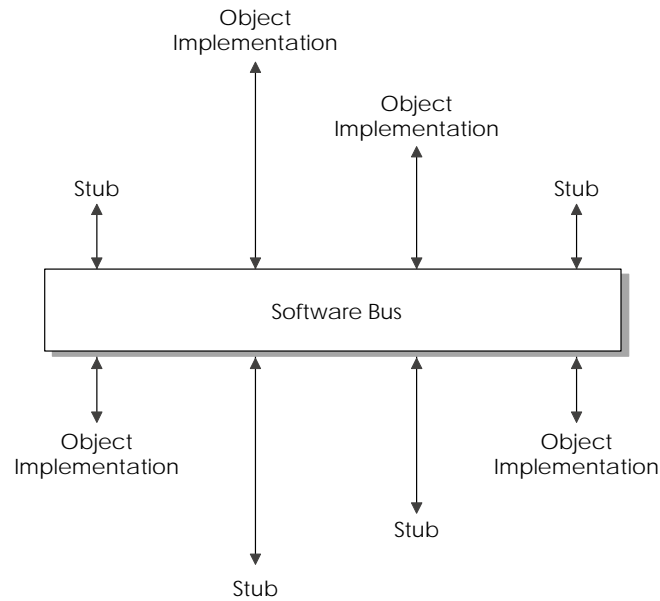
CORBA uses an Interface Definition Language (IDL) to specify the interface for objects. CORBA basically provides a bus to CORBA components.

The CORBA infrastructure consists of four main components:

Object Request Broker (ORB)	Defines the CORBA object bus. Provides middleware functionality so you can use objects in different languages and locations without knowing details about the objects. Each ORB uses specific object naming conventions to locate objects.
Common Object Services	Defines system level object frameworks.
Common Facilities	Defines application frameworks used by business objects.
Application Objects	Defines the business objects and applications used. In OneWorld this is the OneWorld business functions used.



Each ORB has an IDL compiler and an Interface Repository. The interface repository contains metadata that contains the interface specifications for CORBA objects.



The structure of an IDL file includes:

Module	Defines name identifiers.
Interface	Defines a set of methods or operations that can be used with an object.
Operations	Defines services that can be used with objects, for example, parameters and attribute values.
Data Types	Defines accepts attributes, exceptions, and return values for objects.

The basic steps for using CORBA include the following:

- Use IDL to define object classes
- Run the IDL through a language precompiler to process the IDL files and produce skeletons for implementation
- Add implementation code to the skeletons
- Compile the code

This creates information for the interface repository, client IDL stubs, server IDL stubs and object implementations.

CORBA and OneWorld

A OneWorld business function is a logical collection of C functions and their associated data structures grouped together to produce a unit of work.

OneWorld CORBA objects are effectively wrappers around these business functions and data structures. OneWorld CORBA support is based on OneWorld JAVA support.

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

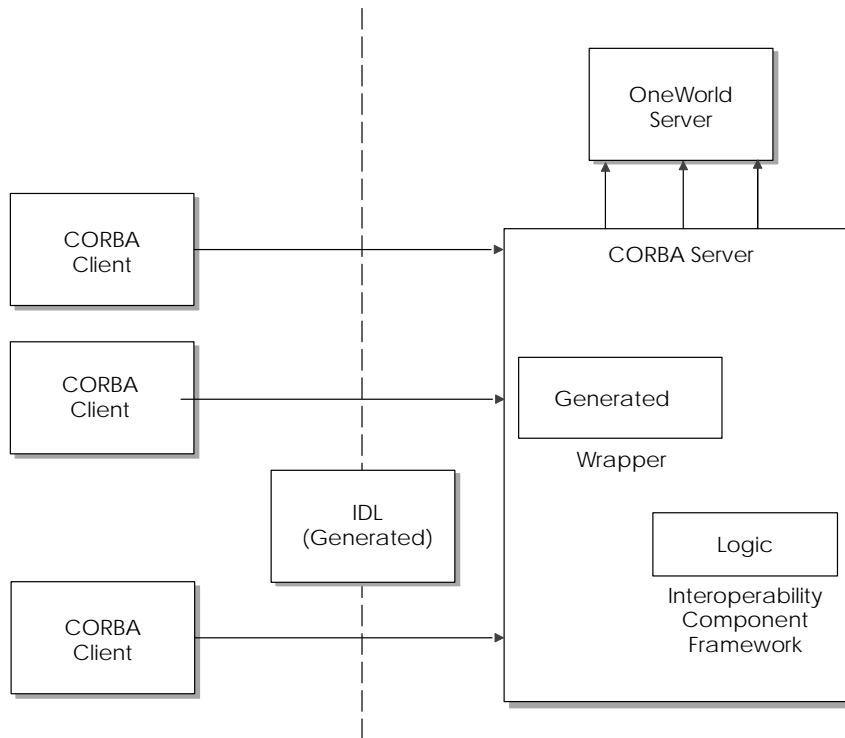
```
interface DS1
{
    A1_getA1();
    void setA1(A1 value);
}
interface DS2
{
    B1_getB1();
    void setB1(B1 value);
}
interface B550001
{
    long foo1(inout DS1 params, in OneWorldInterface owlInterface, in Connector connector, in long
sessionHandle);
    long foo2(inout DS2 params, in OneWorldInterface owlInterface, in Connector connector, in long
sessionHandle);
};
```

Understanding GenCORBA

GenCORBA generates CORBA interfaces for OneWorld business functions. This includes an IDL file and a Java jar file with the stub/skeleton classes generated by the IDL compiler and those implementation classes generated by the built-in CORBAEmitter. The default IDL compiler is OrbixWeb's idl.exe, but the user can always use a different IDL compiler as an option. To compile the Java classes, Kernel.jar, Connector.jar, CorbaConnector.jar, and JDECORBAConnectorOrbixWeb.jar must be deployed to the machine and added to the classpath. These four jar files consist of a group of foundation classes for the CORBA server to interface with the OneWorld Enterprise server. JdeCorbaConnector.idl must be on the machine too.

When you run GenCORBA, you can specify a library, for example CAEC. GenCORBA creates CAEC.idl and CAEC.jar.

The following diagram illustrates CORBA server deployment for the server environment.



Understanding the CORBA Generator (GenCORBA)

The OneWorld CORBA generation tool, GenCORBA, provides access to OneWorld business functions by generating CORBA interfaces for OneWorld business functions. When you use GenCORBA to generate these interfaces you must specify an Object Request Broker (ORB) vendor. Currently J.D. Edwards supports generation capabilities for Orbix and VisiBroker. GenCORBA includes the following components:

- GenCORBA.exe
- Emitter framework
- CORBA emitter.dll

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

Running GenCORBA

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

Syntax

```
GenCORBA [options] [libraries]
```

Example

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

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

```

C:\work>gencorba /?
J.D. Edwards OneWorld(tm) Interoperability Interface Generator
Copyright (C) 1996-2000 J.D. Edwards World Source Company
All Rights Reserved.
Usage: gencorba [options] [libraries]
options:
/?                Display this help information
/Cat <category>  Generate only <category> function wrappers
Supported categories:
/'1/'            - Master Business Functions
/'2/'            - Major Business Functions
/'3/'            - Minor Business Functions
/'-/'            - Uncategorized Business Functions

/Cmd <file>       Process commands from <file>
/Cmd *            Process commands from the console
/Compiler <file>  Use <file> to compile java files
/D name value     Define a macro value
/EnvironmentID <env> Use <env> to log into OneWorld
/ErrFile <file>   Use <file> to record all error messages
/IDL <file>        Use <file> as the IDL compiler
/IDLINCLUDE <flag> Use <flag> for IDL compiler include dir
/IDLOUTPUT <flag>  Use <flag> for IDL compiler output dir
/IDLEXTRA <flags> Pass extra flags to IDL compiler
/IMPLCLASSFORMAT <p> Use <p> as format for base class name of Impls
/ListLibraries    List the libraries in Object Librarian Table
/MsgFile <file>   Use <file> to record all messages
/NoBSFN           Generate wrappers for parametersets only
/Out <path>        Use <path> for all output files
/Password <password> Use <password> to log into OneWorld
/TempOut <path>   Use <path> for all temporary files
/UserID <userid>  Use <userid> to log into OneWorld

Example:
gencorba /Cat 1 /Cat 2 CAEC
- Generates CORBA wrappers for all category 1 and 2 business functions in
the CAEC library

C:\work>_

```

Options

<code>/?</code>	Lists the options available for generation.
<code>/Cat <category></code>	Generates only <category> function wrappers. Supported categories are:
	<code>/'1/'</code> - Master Business Functions
	<code>/'2/'</code> - Major Business Functions
	<code>/'3/'</code> - Minor Business Functions
	<code>/'-/'</code> - Uncategorized Business Functions
<code>/Cmd *</code>	Processes code generation commands from the console. Refer to <i>Understanding iJDEScript</i> for more information.
<code>/Cmd <filename></code>	Processes code generation commands from <filename>. Refer to <i>Understanding iJDEScript</i> for more information.
<code>/Compiler <file></code>	Uses <file> to compile java files.
<code>/D name value</code>	Defines a macro value.
<code>/EnvironmentID <env></code>	Uses <env> to log into OneWorld.
<code>/ErrFile <file></code>	Uses <file> to record all error messages.
<code>/IDL <file></code>	Uses <file> as the IDL compiler.
<code>/IDLINCLUDE <flag></code>	Uses <flag> for IDL compiler include dir.
<code>/IDLOUTPUT <flag></code>	Uses <flag> for IDL compiler output dir.
<code>/IDLEXTRA <flags></code>	Passes extra flags to IDL compiler.

/IMPLCLASSFORMAT <p>	Uses <p> as format for base class name of Impls, for example, ./IMPLCLASSFORMAT “*_ImplBase”.
/ListLibraries	Lists the available libraries that you can use for GenCORBA.
/MsgFile <file>	Provides GenCORBA with the filename to log messages produced by GenCORBA during the generation process, for example, “messages.log”.
/NoBSFN	Tells GenCORBA not to create wrappers for business functions. This option is for generating parameter sets only.
/Out <path>	Provides GenCORBA with the directory (path) in which to place the output files, for example “C:\winnt\system32”.
/Password <password>	Provides GenCORBA with the password with which you wish to log into OneWorld.
/TempOut <path>	Provides GenCORBA with the directory (path) in which to place temporary files needed for the build process, for example, “C:\temp”.
/UserID <userid>	Provides GenCORBA with the username with which you wish to log into OneWorld.
/XMLOnly	Generate only the XML file.

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

```

Command Prompt
D:\B7\System\Bin32>gencorba /ListLibraries
J.D. Edwards OneWorld(tm) Universal Interface Generator      Version B73.3.1
Copyright (C) 1994-1998 J.D. Edwards & Company
All Rights Reserved.
CAEC
CALLBSFN
CAPS
CBUSPART
CCOMVERT
COORE
CCRIN
CCUSTOM
CDBASE
CDDICT
CDESIGN
CDIST
CFIN
CHR
CINSTALL
CINU
CLOC
CLOG
CMFG
CMFG1
CMFGBASE
COBLIB
COBLIB
COPBASE
CRUNTIME
CTOL
CTRAN
CTRANS
CWARE
CWARELOW
JDBTRG1
JDBTRG2
JDBTRG3
JDBTRG4
JDBTRIG
D:\B7\System\Bin32>
    
```

Using GenCORBA Output

For Java applications, Connector.jar, CorbaConnector.jar, JDECORBAConnectorOrbixWeb.jar, and the jar file generated by GenCORBA can be simply deployed to a machine for developing CORBA client applications using OrbixWeb. These jar files must be added to the classpath. These jar files consist of both client side classes and server side classes. An “IDL to Java” IDL compiler can also be run using the jdeCorbaConnector.idl and the IDL file generated by GenCORBA to generate only stub classes for client applications. Any framework component class, such as Connector, or OneWorldInterface is in com.jdedwards.system.corba. Any business function wrapper class is in com.jdedwards.application.corba.LibraryName, for example D0100033 is in com.jdedwards.application.corba.jdeaddressbook.

For a C++ application, or any other kind of application, a specific kind of IDL compiler must be run to generate the stub code from the IDL files.

Any CORBA client application using OrbixWeb must do following:

1. Initialize OrbixWeb for an application
2. Establish a CORBA connection with the CORBA server and return a proxy for the server Connector object. The bind() call causes the OrbixWeb daemon to launch the jdeCORBAServer server, and enables it to accept a remote request. Alternatively, jdeCORBAServer can be configured to run persistently, and the client can obtain the Connector proxy via its stringified object reference.
3. Log in to OneWorld using a valid OneWorld user ID, password, and environment name. The environment must be valid on the OneWorld enterprise server.
4. Get the OneWorldInterface object reference by calling CreateBusinessObject() with the object name CORBA::OneWorldInterface.
5. Get the object reference for the wrapper for the OneWorld business function generated by GenCORBA, for example AddressBook. The object name passed into CreateBusinessObject should be “Library (Java package) Name:Object Name”, such as “JDEAddressBook:AddressBook”.
6. Call CreateXXXParameterSet() or the wrapper object for any data structure XXX.
7. Set the required value in the data structure.
8. Call the business function with the data structure variable as a parameter. Check the return value. Process the data returned by the business function.

Successful = 0

Error = 1

Warning = 2

9. Free objects.
10. Log off OneWorld.

Following is an AddressBook example written in Java using OrbixWeb.

```
import org.omg.CORBA.SystemException;
import org.omg.CORBA.ORB;
import IE.Iona.OrbixWeb._CORBA;
import IE.Iona.OrbixWeb._OrbixWeb;
import OrbixWebDemoManager.*;
import com.jdedwards.system.corba.*;
import com.jdedwards.application.corba.jdeaddressbook.*;

public class abclient
{
    public static void main (String[] args) {
        Connector connectorProxy = null;
        String hostname = null;
        OneWorldInterface ow;
        AddressBook ab;
1.      ORB orb = ORB.init(args, null);
        if (args.length >= 1) {
            hostname = new String(args[0]);
        } else hostname = _OrbixWeb.ORB(orb).myHost ();
        try {
2.          connectorProxy = ConnectorHelper.bind("jdeCORBAServer", hostname);
        }
        catch (org.omg.CORBA.SystemException ex) {
            System.out.println("Exception during bind : " + ex.toString());
            System.exit(1);
        }
        org.omg.CORBA.ObjectHolder obj1 = new org.omg.CORBA.ObjectHolder();
        org.omg.CORBA.ObjectHolder obj2 = new org.omg.CORBA.ObjectHolder();
        D0100033 ds;
        D0100033Holder params = new D0100033Holder();
        int i = 0;
        try {
3.          i = connectorProxy.Login("foo", "bar", "ADEVHPO2W");
        } catch (reject r) {
            String s = r.reason;
            System.exit(1);
        }
        try {
4.          connectorProxy.CreateBusinessObject("CORBA::OneWorldInterface", i, obj1);
        } catch (reject r) {
            String s = r.reason;
            System.exit(1);
        }
        try {
5.          connectorProxy.CreateBusinessObject("JDEAddressBook::AddressBook", i, obj2);
        } catch (reject r) {
            String s = r.reason;
            System.exit(1);
        }
        ow = OneWorldInterfaceHelper.narrow(obj1.value);
        ab = AddressBookHelper.narrow(obj2.value);
6.          ds = ab.CreateGetEffectiveAddressParameterSet();
        params.value = ds;
7.          ds.getAddressNumber().setValue("4242");
        int i = 0;
        try {
8.          i = ab.GetEffectiveAddress(params, ow, connectorProxy, i);
        } catch (reject e) { System.out.println(e.reason); }
        if (i != 1) {
            String alphaname = ds.getNameAlpha();
            System.out.println("Name:" + alphaname);
        }
    }
}
```

```

        if (i == 2) {
            for ( int j = 0; j < ow.GetWarningCount(); j++) {
                System.out.println("warning" + j + ":" + ow.GetWarningAt(j));
            }
        }
    } else {
        for (int j = 0; j < ow.GetErrorCount(); j++) {
            System.out.println("error" + j + ":" + ow.GetErrorAt(j));
        }
    }
9   connectorProxy.FreeBusinessObject(ab);
    connectorProxy.FreeBusinessObject(ds);
10  connectorProxy.FreeBusinessObject(ow);
    connectorProxy.Logoff(l);
}

```

Working with JDEDate and JDEMathNumeric CORBA Objects

Parameter sets may have JDEDate and JDEMathNumeric data members. The parameter set itself is a CORBA Object, and it is important to realize that any JDEDate or JDEMathNumeric members of the parameter set are themselves CORBA objects. The get() method for these data members returns a reference to the CORBA Object.

The IDL for JDEDate and JDEMathNumeric is as follows:

```

interface JDEDate
{
    short getYear();
    void setYear(in short year);

    short getMonth();
    void setMonth(in short month);

    short getDay();
    void setDay(in short day);
};

interface JDEMathNumeric
{
    void reset();

    string getValue();
    void setValue(in string value) raises(reject);

    string getCurrencyCode();
    void setCurrencyCode(in string value) raises(reject);

    long getCurrencyDecimals();
    void setCurrencyDecimals(in long value) raises(reject);
};

```

To get/set the members of a JDEDate or JDEMathNumeric, first get() the object from its parameterset. Calling the set() methods of the object operates directly on the object in the parameterset. Do not call FreeBusinessObject() using the

JDEDate or JDEMathNumeric as input. These objects are freed when FreeBusinessObject is called on their parameterset object.

Setting Up a OneWorld Client Environment for GenCORBA

There are several steps you must take to set up a OneWorld client environment for GenCORBA. You should make sure the following are set up appropriately:

- PATH environment variable
- CLASSPATH environment variable

PATH

<bin directory for OrbixWeb>

Example: c:\iona\bin

<bin directory for JDK>

Example: c:\jdk1.2.2\bin

CLASSPATH

<Directory where JDEdwards OneWorld is located>\System\classes\Kernel.jar

<Directory where JDEdwards OneWorld is located>\System\classes\Connector.jar

<Directory where JDEdwards OneWorld is located>\System\classes\CORBAConnector.jar

<Directory where JDEdwards OneWorld is located>\System\classes\JDECORBAConnectorOrbixWeb.jar

Setting Up an Environment for CORBA

GenCORBA is designed to work with a variety of CORBA vendors and IDL compilers. While the different IDL compilers all generate CORBA-compliant stubs and skeletons, they all generate slightly different code in the stubs and skeletons. Furthermore, stubs and skeletons generated by an IDL compiler of one vendor will not be compatible with the ORB of another. The same is true for different versions of OrbixWeb. Stubs and skeletons generated with the IDL compiler for version 3.1 are not compatible with the ORB for version 3.2. For this reason, we do not ship any stubs or skeletons.

To set up GenCORBA, you must first compile JDECORBAConnector.idl with your particular IDL compiler. You create a jar file named JDECORBAConnectorOrbixWeb.jar:

1. Create a temporary directory.
2. Copy JDECORBAConnector.idl from the system\include directory of your OneWorld install to the temporary directory.
3. Change directories to the temporary directory.
4. Run the OrbixWeb idl compiler: `idlj JDECORBAConnector.idl`.
5. Change directories to `java_output\com\jdedwards\system\corba`.
6. Compile the .java files: `javac -classpath c:\Iona\lib\OrbixWeb.jar *.java`.
7. Change directories back to the temporary directory: `cd ..\..\..\..\`.
8. Create the jar file: `jar cvf JDECORBAConnectorOrbixWeb.jar -C java_output com`
9. Copy the file to the system\classes directory of your OneWorld install.

You must perform these steps initially, to set up GenCORBA, and also whenever you upgrade to a later version of OrbixWeb.

Building and Deploying a CORBA Server

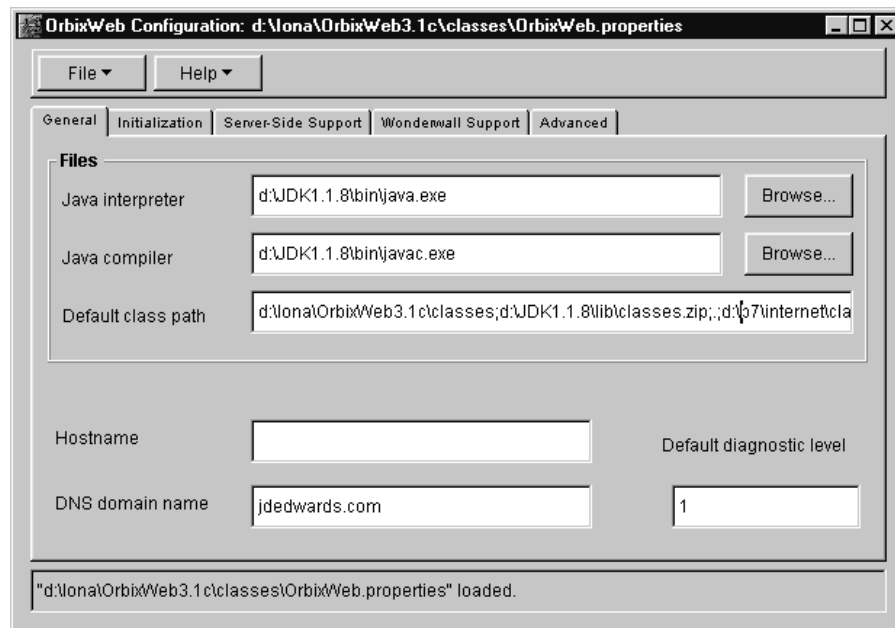
1. Run GenCORBA to generate the IDL file and the jar file.
2. Deploy the Jar file generated in step 1 to the server machine.
3. Install Kernel.jar, Connector.jar, CorbaConnector.jar, JDECORBAConnectorOrbixWeb.jar, and jdeinterop.ini to the server machine.
4. Get the CORBA server ready.

Refer to Setting up the CORBA server for the details.

Setting up a CORBA Server

The CORBA server can be on the same machine as the OneWorld enterprise server or on a completely separate machine. However, OrbixWeb and JDK must be installed. Complete the following steps to set up a CORBA Server.

1. Deploy the jar files (Connector.jar, Kernel.jar, CorbaConnector.jar, JDECORBAConnectorOrbixWeb.jar) and jdeinterop.ini to the CORBA server.
2. Add the jar files to the class path.
3. The CORBA Server Administrator must run the OrbixWeb configuration tool to add all jar files to the default class path. On NT, go to Start/Program/OrbixWeb/Configuration Tool. The following illustration is from OrbixWeb3.1. The interface looks slightly different for OrbixWeb3.2. On Unix, such as Solaris, run ConfigurationExplorer.sh under iona/bin.



4. Create a repository directory for the generated business object libraries (JAR files).
5. Deploy any JAR files generated by GenCORBA to the repository directory on the CORBA server.
6. Set an environment variable, such as JDECORBA_HOME, and point it to the place where jdeinterop.ini was deployed.
7. Start OrbixWeb Daemon and execute the following command:

```
putit -j [-jdk2] jdeCORBAServer " -Dconfig_file=%JDECORBA_HOME%\jdeinterop.ini -
```



```
Dcom.jdedwards.system.corba.ORBWrapperClass=com.jdedwards.system.corba.OrbixORB  
wrapper com.jdedwards.system.corba.jdeCORBAServer”
```

Use the `-jdk2` option if you are using OrbixWeb3.2 and JDK1.2 or higher.

You can also run the following command:

```
putit -persistent jdeCORBAServer
```

You can then start the server yourself using the following command:

```
java -Dconfig_file=%JDECORBA_HOME%\jdeinterop.ini -Dcom.  
jdedwards.system.corba.ORBWrapperClass=com.jdedwards.system.corba.OrbixORBWrapper  
[-Dorg.omg.CORBA.ORBClass=IE.Iona.OrbixWeb.CORBA.ORB -Dorg.  
omg.CORBA.ORBSingletonClass=IE.Iona.OrbixWeb.CORBA.singletonORB]  
com.jdedwards.system.corba.jdeCORBAServer
```

The `-D...ORBClass` and `-D...ORBSingletonClass` are only necessary if you are using OrbixWeb3.2, and JDK1.2 or higher.

iJDEScript

GenCOM, GenCORBA, 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 OneWorld object aliases for more meaningful names.
- Select OneWorld 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, GenCORBA, 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%,

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, 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, 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 OneWorld 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 OneWorld library, such as CAEC or CALLBSFN, into the current library definition. Each business function group in OneWorld 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
```

Syntax for CORBA

```
interface interface
```

CORBA Example

```
interface 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 into OneWorld. If *user*, *password*, and *environment* are not specified, the user is prompted for the information.

Syntax

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

Example

```
login me mypassword demo
```

Logout Command

The logout command tells the generator to log out of OneWorld.

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%,

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
```


jdeinterop.ini

The jdeinterop.ini file includes some settings the server may 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]. Sections are listed in the order they are found in the software.

[JDENET]

Setting	Typical Value	Purpose
EnterpriseServerTimeout=	90000	The timeout value for a request to the enterprise server.
maxPoolSize=	30	The JDENET socket connection pool size.

[SERVER]

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



[LOGS]

Setting	Typical Value	Purpose
log=	c:\jas.log	This is the file used by interoperability components to log significant information.
debuglog=	c:\jasdebug.log	Location of debug log file.
Debug=	FALSE	Turn on/off debug tracing by the thinnet component on the CORBA server and Pure Java. TRUE sets tracing on. FALSE sets tracing off.

[DEBUG]

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	<p>This defines the level of tracing provided by the COM Connector and the Callobject component in the specified log file, in the COM server only.</p> <ul style="list-style-type: none"> 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) <p>Note:</p> <ol style="list-style-type: none"> 1. The odd values are reserved for future levels to be added. 2. You typically do not need tracing on by default. However, tracing is useful for debugging.
NetTraceLevel=	0	<p>This defines the level of tracing provided by the thinnest component in the specified log file, in the COM server only.</p> <ul style="list-style-type: none"> 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. <p>You typically do not need tracing on by default. However, tracing is useful for debugging.</p>

[INTEROP]

Setting	Typical Value	Purpose
enterpriseServer=	JDED	The OneWorld Enterprise server.
port=	6010	The port number of the OneWorld Enterprise server.
inactive_timeout=	1200000	The timeout value for a transaction in auto commit mode. If the user is inactive for this amount of time (in milliseconds), the interop server will log off the user.

Interoperability

manual_timeout=	300000	The timeout value for a transaction in manual commit mode.
Repository	c:\JDEdwards\Interop\repository	Points to the location of the repository directory containing business object libraries (generated JAR files).

[CORBA]

Setting	Typical Value	Purpose
Multithread=	1	Set to 1 for multithread support for CORBA
Objects=	CORBA::Connector;CORBA::OneWorldVersion	The objects for the CORBA server to create at startup. Also replaces the -DIORFILENAME = command line option, for example CORBA::Connector=connector.ior

XML



XML

This section describes some of the J.D. Edwards features available to help you implement an XML solution. It includes the following topics:

- Understanding XML
- Working with XML CallObject
- Creating an XML template
- Setting the jde.ini file for XML



Understanding XML

Extensible Markup Language (XML) is similar to Hypertext Markup Language (HTML). It is an extensible structured language that allows you to define how data is handled. This information may include how data is stored, transmitted, or processed. XML separates content from the format of the content. This supports using information from one application so that it can be used in another application. For example, one company's application can use customer information, such as name, address, phone, and subscription options for producing bills for cable television services, while another company's application can use that same information for determining which commercials to show in different geographical areas.

The J.D. Edwards XML solution supports well-formed XML documents. It also supports UTF8 and UTF16 Unicode standards for inbound information and UTF8 standards for outbound information.

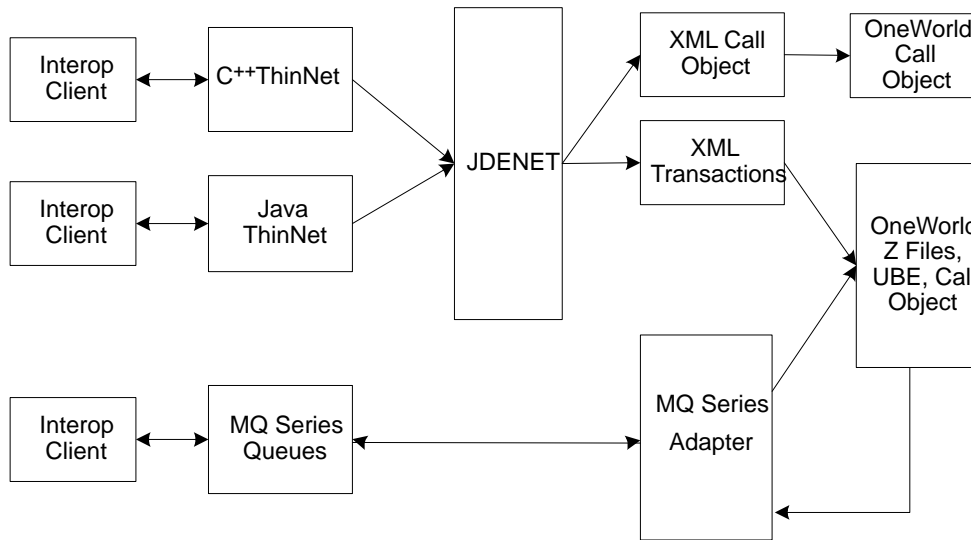
The document object model (DOM) stores the data structure of an XML document. It uses a tree structure that has a root document and a tree of elements and attributes.

The following diagram illustrates the typical structure for an XML document.

```
<Elem>
  <SubElem name=" " version " ">
  </SubElem>
</Elem>
```

XML and OneWorld

The following diagram illustrates the flow for an XML transaction.



First an interoperability client sends an XML document to OneWorld. The document format is defined by J.D. Edwards.

The client uses APIs defined by either C++ or Java ThinNet to send the XML document to JDENET. C++ and Java ThinNet use a multi-threaded architecture to do load balancing and to manage multiple XML documents simultaneously.

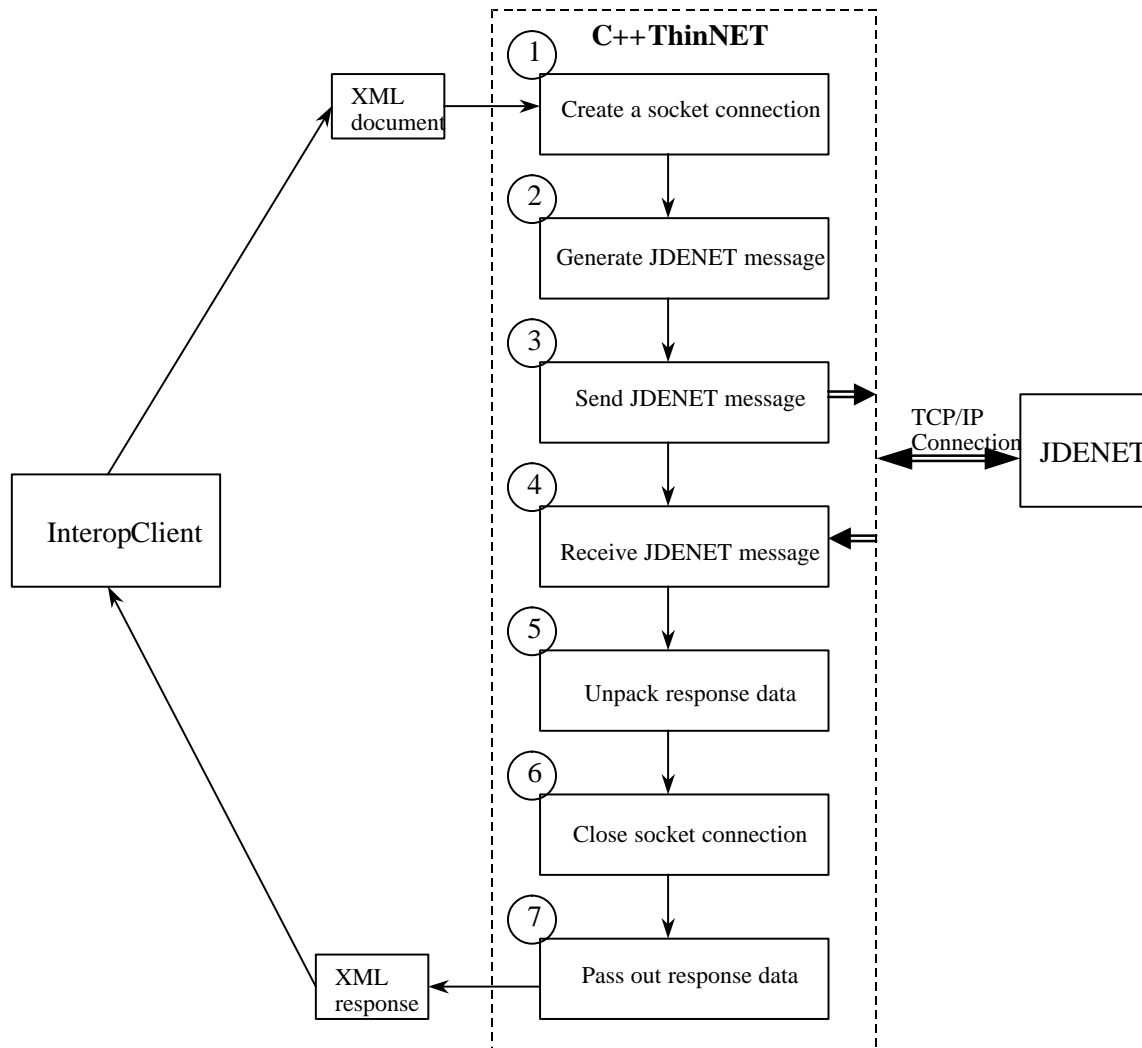
Once the documents reach JDENET, the next step in the process depends on what document type is being sent. There are two XML document formats: CallObject and XML Transaction APIs. CallObject is used for synchronous requests only. XML transactions are usually used for asynchronous requests. All XML documents contain an element that identifies the request as a Call Object or a Transaction. Based on the document type, JDENET decides which kernel to send the document to.

You can also use MQ Series and input queues to send the message to OneWorld using the OneWorld Adapter for MQSeries. The server must be running so that JDENET is running. For more information refer to the *Asynchronous Messaging Adapter Programmer's Guide*.

ThinNet

The following information describes the process flow for C++ ThinNet used in XML CallObject.

Refer to the Online API documentation for information about specific APIs.

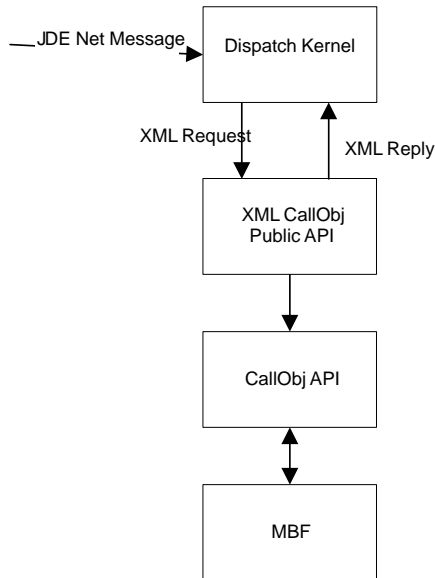


1. An interoperability client sends an XML document to C++ ThinNET to initiate an XML request. ThinNET creates a socket connection to JDENET on a OneWorld server. Each XML request has its own unique network connection to the server.
2. The XML document is packed into a data segment of a JDENET message.
3. ThinNET sends the message out to JDENET on the server.
4. ThinNET blocks in receiving a response message from the server. A timeout error occurs if no response is received from the server within the user-specified timeout in seconds.
5. The response JDENET message is unpacked and the response data is extracted.
6. Network connection is closed for this XML request.
7. Response data is passed out to the interoperability client.

The process flow for Java ThinNet is the same as the flow for C++ ThinNet. An API is used to submit XML documents to the server.

Inbound and Outbound Synchronous XML CallObject

The following diagram depicts the synchronous process using XML CallObject.



Inbound Synchronous Process Flow

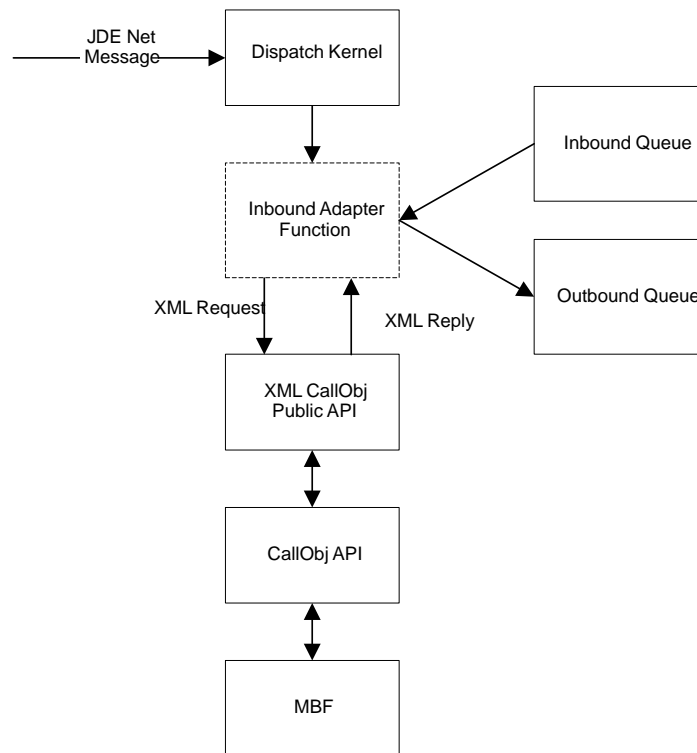
Typically, inbound or outbound XML processing using XML CallObject follows a flow similar to the following:

- The client sends a message through JDE Net to the appropriate kernel.
- The XML document is passed into the `jdeXMLCallObject` API.
- The OneWorld server processes the message by parsing the XML document. The Session Manager also validates user and password.
- Each requested business function is called separately or within requested transaction boundaries until all calls are processed.
- Transactions are added to the OneWorld database.
- Output data and error messages are merged with the data from the input XML document and a new response document is created.

Inbound Asynchronous XML Transactions

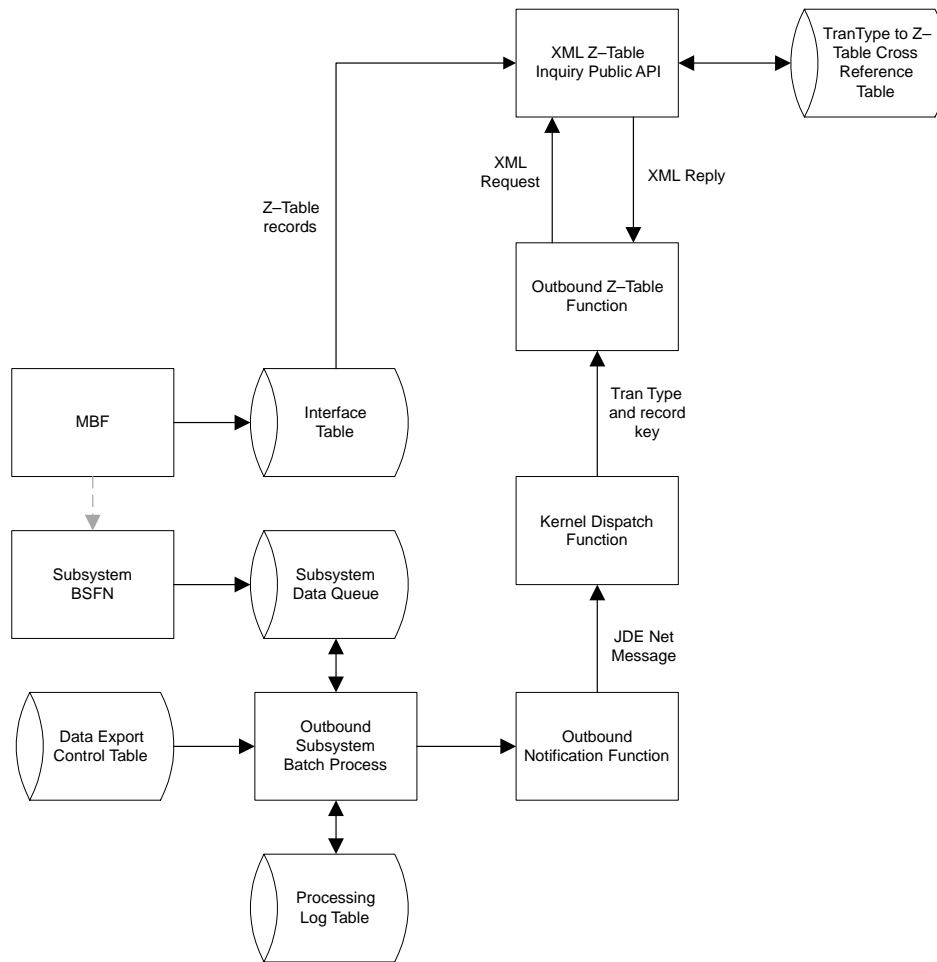
The inbound asynchronous XML transaction process uses XML Call Object methodology for processing inbound transactions, such as a sales order. The Inbound Adapter Function is used for transactions using an adapter, such as the OneWorld Adapter for MQSeries, only.

The following diagram depicts the inbound asynchronous process.



Outbound Asynchronous XML Transactions

The following diagram depicts the outbound process, which uses the Z-Table asynchronous methodology for passing a variety of information to the attached system.



Outbound Asynchronous Process Flow

Typically, outbound processing for XML transactions follows a flow similar to the following:

- An outbound message is triggered by an event, for example entry of a sales order.
- Subsystem processing starts processing the transaction and calls the outbound notification function.

Refer to the Interoperability guide for more information about subsystem processing.

- The client can retrieve the transaction information by sending an XML request with an action attribute of 'TransactionInfo' and a type attribute of 'transaction type'.

Working with XML CallObject

The following information explains how CallObject works and details the steps involved with using XML CallObject.

Establish Session

A session must be established. This step is addressed by the session attribute of the standard jdeRequest element. If the session attribute is an empty string, this indicates that the user wants a session started. On the server the SessionManager singleton class creates a new instance of a Session object given the user name, password and environment name.

```
<?xml version='1.0' ?>
<jdeRequest type='callmethod' user='steve' pwd='xyz'
environment='prod' session='' sessionidle='1800'>
...
</jdeRequest>
```

Expire Session

Session expiration is addressed by the sessionidle attribute of the standard jdeRequest element. This attribute, when given on a session creation request, specifies the amount of time in seconds that this session is allowed to be idle. If the SessionManager determines that a session has not had any requests processed in this amount of time, it terminates the session and frees all associated resources. The session idle default is 30 minutes. You can use a jde.ini file setting to change the default session idle time.

```
<?xml version='1.0' ?>
<jdeRequest type='callmethod' user='steve' pwd='xyz'
environment='prod' session='' sessionidle='1800'>
...
</jdeRequest>
```

Call Object

Tags are used to call business functions on the server.

```
<?xml version='1.0' ?>
<jdeRequest type='callmethod' user='steve' pwd='xyz' environ-
ment='prod' >
<callMethod name='myfunc' app='P42101' >
<params>
<param name='CostCtr' >      1001</param>
<param name='ExpDate' >1999/10/31</param>
<param name='Quantity' >12</param>
</params>
</callMethod>
</jdeRequest>
```

The callMethod element details which function to call and in what context it is being called. The name attribute specifies which business function to call and the app attribute allows the business function to know “who” is calling it.

The params and param elements define the data structure of the business function. Each param element describes one data structure member. The caller is only required to give the name attribute.

If no param element value is given for an input data structure member, then the value will be treated as if it were NULL or zero.

Explicit Transaction

Explicit DB transactions are supported by another element, the startTransaction tag. This element specifies whether transactions are going to be manual or automatically committed. The startTransaction element is an empty element, meaning that all of its information is in the attributes.

```
<?xml version='1.0' ?>
<jdeRequest type='callmethod' user='steve' pwd='xyz' environ-
ment='prod' session='' >
<startTransaction trans='t1' type='manual' />
</jdeRequest>
```

Implicit Transaction

A CallObject request is included in a transaction set when the name of a transaction set is referenced in its trans attribute. Implicit start transactions can be

included in the CallObject request by specifying the name of a transaction set that has not previously been created. If it is an implicit start, the transaction set will be a manual commit set.

```
<?xml version='1.0' ?>
<jdeRequest type='callmethod' user='steve' pwd='xyz' environ-
ment='prod' session=''>
<callMethod name='myfunc' app='P42101' trans='t1'>
<params>
<param name='CostCtr'>          1001</param>
</params>
</callMethod>
</jdeRequest>
```

Prepare/Commit/Rollback

Manual transaction sets can be committed or rolled back. As part of a two phase commit, they can be prepared to commit. These requests to the database are made using the endTransaction element. The transaction set is identified by the trans attribute and the action attribute indicates the action to take on the transaction set. The value can be 'prepare', 'commit', or 'rollback'. This element is always an empty element, as shown by the forward slash.

NOTE: If startTransaction and endTransaction are in separate documents, one of the following should occur:

- The session attribute is not sent in the second document. In this case the user, pwd and environment are used to match the previous session.
- The session number from the response of the first document is sent in the session attribute of the documents that are associated with the same transaction.

It is recommended to manage the session ID when doing manual commits, and terminate the session after the transaction is complete.

```
<?xml version='1.0' ?>
<jdeRequest type='callmethod' user='steve' pwd='xyz' environ-
ment='prod' session=''>
<endTransaction trans='t1' action='commit' />
</jdeRequest>
```

Terminate Session

Session termination is done by submitting an XML document to explicitly terminate the session. The caller must specify the session to be terminated in the `jdeRequest` element tag.

```
<?xml version='1.0' ?>
<jdeRequest type='callmethod' user='steve' pwd='xyz' environ-
ment='prod' session=5665.931961929.454' >
<endSession/>
</jdeRequest>
```

Call Object Error Handling

System errors on a call object are reported in the `returnCode` element. The numeric code is returned in the `code` attribute and the corresponding text is returned as a child text node of the `returnCode` element. The standard `jdeCallObject` return codes are used for the `code` attribute.

```
<?xml version='1.0' ?>
<jdeResponse type='callmethod' user='steve' pwd='xyz' environ-
ment='prod' session='' >
<callMethod name='myfunc' app='P42101' trans='t1' >
<params>
<param name='CostCtr' >      1001</param>
</params>
<returnCode code='0' >Success</returnCode>
</callMethod>
</jdeResponse>
```

BSFN Error Handling

Business function error handling includes using error text, errors during multiple requests, and errors that occur during call method requests.

Error Text

Business function error messages are returned in the `errors` element. Within that element there can be zero or more error elements, containing a `code` attribute for the error code and a child text node containing the error text. The `name` attribute describes which param element the error refers to.

```

<?xml version='1.0' ?>
<jdeResponse type='callmethod' user='steve' pwd='xyz' environ-
ment='prod' session=''>
<callMethod name='myfunc' app='P42101' trans='t1'>
<params>
<param name='CostCtr'>          1001</param>
</params>
<returnCode code='2'>Errors</returnCode>
<errors>
      <error code='192' name='CostCtr'>Cost Center not valid</er-
ror>
</errors>
</callMethod>
</jdeResponse>

```

Multiple Requests per Document

Multiple requests can be included in the XML document. By default, requests are not run if there have been any errors on previous requests. If a request should be run even if errors have occurred, then the default behavior can be overridden by using the `runOnError` attribute on the request with a value of `yes`.

```

<?xml version='1.0' ?>
<jdeRequest type='callmethod' user='steve' pwd='xyz' environ-
ment='prod' session=''>
<callMethod name='myfunc' app='P42101' trans='t1' runOnEr-
ror='yes'>
<params>
<param name='CostCtr'>          1001</param>
</params>
</callMethod>
</jdeRequest>

```

On Error Handling

If an error occurs on a `callMethod` request you can add an `onError` element to the request to take some action, such as calling another business function, to clean up resources. The `onError` tag can specify an `abort` attribute that specifies if all subsequent requests should be skipped. The allowed values are `'yes'` or `'no'`. A `'global'` `onError` tag can be specified as a child of the `jdeRequest` tag, that will be executed if there were errors encountered and no other `onError` tag with

abort='yes' was executed. The global onError tag should be the last request in the document.

```
<?xml version='1.0' ?>
<jdeRequest type='callmethod' user='steve' pwd='xyz' environ-
ment='prod' session=''>
<callMethod name='myfunc' app='P42101' trans='t1' runOnEr-
ror='yes'>
<params>
<param name='CostCtr'          1001</param>
</params>
<onError abort='no'>
      <endTransaction trans='t1' action='rollback' />
</onError>
</callMethod>
</jdeRequest>
```

ID/IDREF Support

ID type attributes uniquely identify, by a string value, elements in a XML document. IDREF attributes allow other elements to reference the specified element. An IDREF attribute must not be used in a document before the ID it references is defined.

A param element can specify an id attribute so that its output value from the callMethod request will be saved and referred to later in another param element by an idref attribute. If a param element contains an idref attribute, the value of the given parameter is used as the input value for the param element. For example, the output value from referenced parameter is used instead of the value in the XML.


```

<?xml version='1.0' ?>
<jdeRequest type='callmethod' user='steve' pwd='xyz' environ-
ment='prod' session=''>
<callMethod name='myfunc' app='P42101' trans='t1' runOnEr-
ror='yes'>
<params>
<param name='CostCtr'>          1001</param>
<param name='Company1' id='c1'></param>
<param name='Company2' id='c2'></param>
</params>
</callMethod>
<callMethod name='myfunc2' app='P42101' trans='t1' runOnEr-
ror='yes'>
<params>
<param name='Company1' idref='c1'></param>
</params>
<returnParams><param idref='c2' /></returnParams>
</callMethod>
</jdeRequest>

```

You can specify a special request tag called `returnParams` that can contain one or more param elements. If the param elements contain *idref* attributes, then the referenced values are copied into the response.

Return NULL Values

By default if a parameter was not specified in the request document, it will not be returned in the response document unless its value is non-blank or non-zero. This behavior can be modified by specifying the *returnNullData* attribute on the `callMethod` element with a value of 'yes'.

```
<?xml version='1.0' ?>
<jdeRequest type='callmethod' user='' pwd='' environment='prod'
session=''>
<callMethod name='myfunc' app='P42101' returnNullData='yes'>
<params>
    <param name='CostCtr' ></param>
    <param name='ExpDate' ></param>
    <param name='Quantity' ></param>
</params>
</callMethod>
</jdeRequest>
```

Enabling Outbound Z-Table Processes

The Z-table process is used for XML transaction document types. You use the *OutboundZTable* function to send a message from an outbound Z-Table to an asynchronous messaging queue. The function is invoked from the kernel dispatch function, which then sends the net message data that contains the parameters from the Z-Table subsystem UBE.

```
Void OutboundZTableMessageAdapter(MsgData *pMsgData)
```

These parameters define the records and the transaction type that will be used to cross reference the tables that contain the data to populate the message to be sent to the message queue. The asynchronous messaging-specific Outbound Z-Table Adapter parses the net message data and calls the jde XML Z-Table Inquiry API to fetch the records from the Z-Table and format the results into an XML string.

You must setup OneWorld to initiate the outbound Z-Table process. The format of the outbound Z-Table message retains an XML based format.

Outbound Notification

The business function sends an outbound notification message to JDENET. This function is called by the standard generic Outbound Subsystem batch process UBE and provides notification that records have been placed in the Z-Tables.

This function passes the key fields for a record in the OneWorld Outbound Transaction Z-Tables. With these keys, you can process the information from the database record into a message queue.

```
void MessageNotificationName(char *szUserID, char
*szBatchNumber, char *szTransactionNumber, double
mnLineNumber, char *szTransactionType, char
*szDocumentType, double mnSequenceNumber )
```

Following is the required parameter list. All parameters are input:

User Id – 11 characters

Batch Number - 16 characters

Transaction Number - 23 characters

Line Number - double

Transaction Type - 9 characters

Document Type - 3 characters

Sequence Number - double

This information is sent on in a JDENET message with the following message packets:

Environment name – use JDE APIs to retrieve environment from the Subsystem batch process.

User Id – key to Z-Table record

Batch Number – key to Z-Table record

Transaction Number – key to Z-Table record

Line Number – key to Z-Table record

Transaction Type – tie to which Z-Table

Document Type – (optional)

Sequence Number – (optional)

The key information in the Net message packets is used to retrieve the Z-file record from the table. The transaction type allows the function to be generic and process other transactions in the future. The transaction type maps to the Flat File Cross Reference Table (F47002) to determine the Z-Tables.

XML Z-Table Inquiry API

The XML Z-Table Inquiry API (jdeRetrieveTransactionInfo) receives an XML string that includes the table record key and returns an XML string for outbound.

This function is called from the Asynchronous Messaging Specific Outbound Z-Table. It parses the XML string and, based on the transaction type, goes to the F47002 (Flat File Cross Reference Table) to determine from which Z-Table to fetch records. F47002 has a record for each table associated with the transaction type. Using JDB database APIs, XML Z-Table Inquiry then uses the index found in the XML string to fetch records from the Z-Table and returns the results in an XML string.

Refer to the online Interoperability Interface Tables for more information about specific Z-Tables.

XML Transaction Information Request

The XML transaction information request is created by the outbound function and sent to the XML transaction API.

Example: Outbound Order Status XML Request & Response Format

```
"This format will return all columns for the sales order header (F4201Z1) and detail lines (F4211Z1). "
```

```
<?xml version='1.0' ?>
<jdeRequest type='trans' user='user' pwd='password' environ-
ment='environment' session='
` sessionidle='300'>
  <transaction action='transactionInfo' type='JDESOUT'>
    <key>
      <column name='EdiUserId'>value</column>
      <column name='EdiBatchNumber'>value</column>
      <column name='EdiTransactNumber'>value</column>
    </key>
  </transaction>
</jdeRequest>
```

```
<?xml version='1.0' encoding='utf-8' ?>
<jdeResponse type='trans' user='user' session='session1' environ-
ment='env'>
  <transaction type='JDES00UT' action='transactionInfo'>
    <returnCode code='0'>XML Request OK</returnCode>
    <key>
      <column name='EdiUserId'></column>
      <column name='EdiBatchNumber'></column>
      <column name='EdiTransactNumber'></column>
    </key>
    <table name='F4201Z1' type='header'>
      <column name='EdiUserId'></column>
      <column name='EdiBatchNumber'></column>
    </table>
    <table name='F4211Z1' type='detail'>
      <column name='EdiUserId'></column>
      <column name='EdiBatchNumber'></column>
    </table>
    <table name='F49211Z1' type='additionalHeader'>
      <WARNING>No record found</WARNING>
    </table>
  </transaction>
</jdeResponse>
```


Creating an XML Template

You can create your own custom XML documents. To create your own custom XML documents:

- Use the business function documentation to help you find the specific business functions you want to use. Refer to *Business Function Documentation* in this guide for more information about generating documentation and finding information about specific business functions.
- Create an XML template.

XML CallObject Templates

You create an XML template to make CallObject calls. Refer to *Business Function Documentation* for documentation information about specific business functions.

The caller can request an XML Template for a given business function. This template is an XML document that is a *callMethod* request with information about all the function parameters, but with no data values filled in. The *user*, *pwd* and *session* attribute values are blanked out so that the caller can cache the response for later use.

This request is an exception to the convention that a *jdeRequest* returns a *jdeResponse* to the caller. Although this makes it easier to directly use the template as a *callMethod* request, this is only true if the request is the only request in the document. Refer to the online APIs documentation for information about the jdeXMLRequest API.

The following example illustrates a request.

```
<?xml version='1.0' ?>
<jdeRequest type='callmethod' user='steve' pwd='xyz' en-
viron-ment='prod' session=''>
<callMethodTemplate name='myfunc' app='P42101' />
</jdeRequest>
```

The following example illustrates a response to a request. This response can then be filled in with the appropriate information and sent back as a request.

```
<?xml version='1.0' ?>
<jdeRequest type='callmethod' user='' pwd='' environ-
ment='prod' session=''>
<callMethod name='myfunc' app='P42101'>
<params>
    <param name='CostCtr'></param>
    <param name='ExpDate'></param>
    <param name='Quantity'></param>
</params>
</callMethod>
</jdeRequest>
```


Setting the jde.ini File for XML

The jde.ini settings for the XML call object kernel are as follows:

```
[JDENET_KERNEL_DEF6]
krnlName=CALL OBJECT KERNEL
dispatchDLLName=XMLCallObj.dll
dispatchDLLFunction=_XMLCallObjectDispatch@28
maxNumberOfProcesses=1
numberOfAutoStartProcesses=1
```

The jde.ini settings for the XML transactions kernel are as follows:

```
[JDENET_KERNEL_DEF15]
krnlName=XML TRANSACTION KERNEL
dispatchDLLName=XMLTransactions.dll
dispatchDLLFunction=_XMLTransactionDispatch@28
maxNumberOfProcesses=1
numberOfAutoStartProcesses=1
```

The @28 and the _ in the example above are for NT only.

Refer to the following table for different .dll extensions for other platforms.

	Call Object dispatch DLLName=	XML Trans dispatch DLLName=
AS400	XMLCALLOBJ	XMLTRANS
HP9000B	libxmlcallobj.sl	libxmltransactions.sl
SUN or RS6000	libxmlcallobj.so	libxmltransactions.so

APIs



OneWorld APIs Interoperability Models

This section describes the available OneWorld APIs interoperability models. It includes a conceptual discussion of each model followed by high-level steps for implementation. These steps serve as a task checklist and include an explanation of how the tasks fit together. Details for performing each step are included in the *Detailed Tasks* section. The detailed tasks are grouped into the following categories:

- Detailed tasks for OneWorld native APIs
- Detailed tasks for OneWorld operations
- Detailed tasks for custom programming

You can also use a specific technology with the OneWorld interoperability models. There are specific tasks you complete for each model and technology combination. There are guidelines you can follow to help you determine which model and technology combination to use. You can implement:

- Synchronous transactions into OneWorld
- Asynchronous transactions into OneWorld
- Batch transactions into OneWorld
- Synchronous transactions from OneWorld
- Asynchronous transactions from OneWorld
- Batch transactions from OneWorld

Each chapter in this section includes:

- A diagram of the model for a specific transaction type
- Generic steps to implement the transaction

Processing Modes

There are several available processing modes you can use for transactions into and from OneWorld. The following information describes these modes to help you decide which one best suits your needs.



Synchronous

Synchronous processing implies that you are making a real-time direct call to OneWorld objects. You establish a connection to OneWorld and make calls directly to OneWorld APIs or business functions. The OneWorld object performs its tasks while your calling program waits. Results are immediately available upon completion of the call. You typically use this method for interactive applications requiring immediate user feedback.

Asynchronous

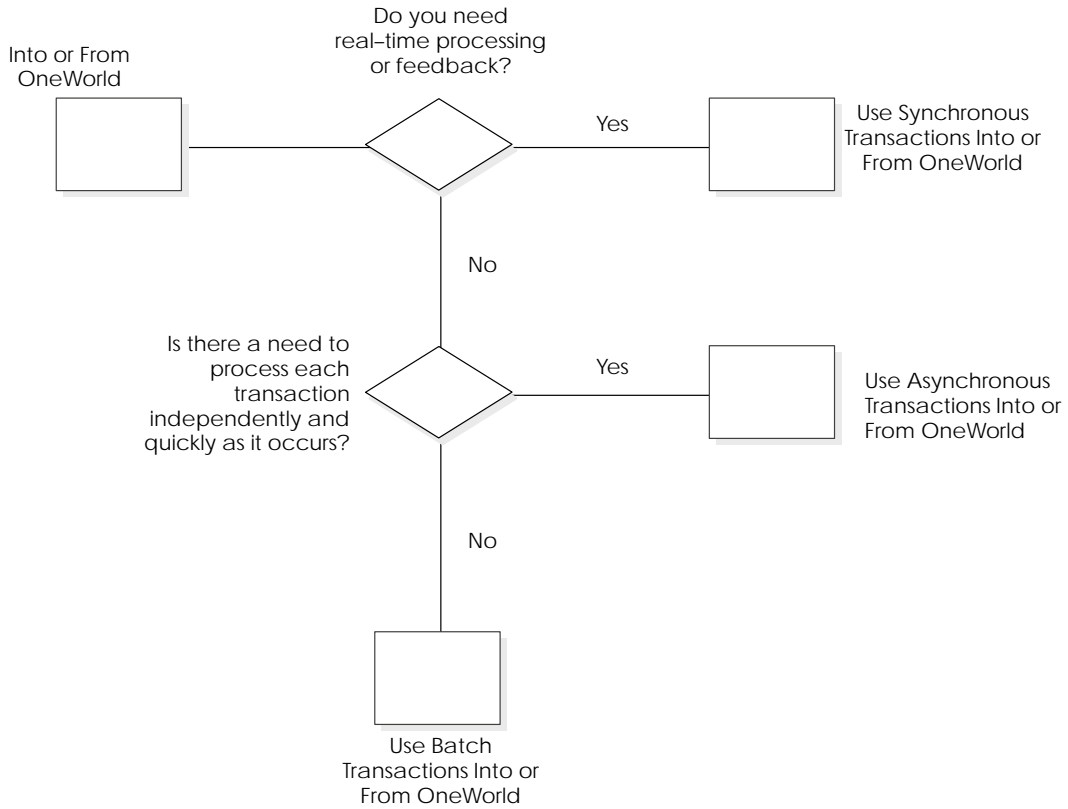
Asynchronous processing allows an application to submit transactions or requests to another application one at a time, but the requests are queued up and processed in the background environment without being directly connected with the calling program. This allows an application to submit a request and immediately continue processing without waiting for a result. Results are returned through a separate process when the transaction is complete. You typically use this method when real-time feedback is not required, although fast transaction response time is still important.

Batch

Batch processing allows you to save transactions over a period of time and then run a periodic process that processes the entire group of transactions at once. You typically use this method for large groups of transactions that must be transferred from one system to another on a periodic basis. Batch processes are often started from a menu.

Choosing a Processing Mode

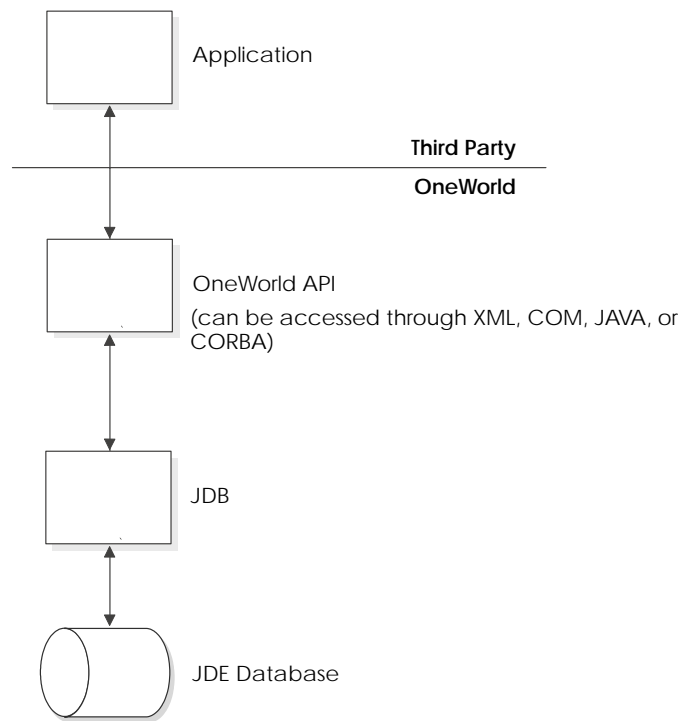
The following illustration shows a decision flowchart to help you choose a processing mode for transactions into and from OneWorld.



Implementing Synchronous Transactions Into OneWorld

Synchronous transactions into OneWorld allow you to communicate directly with OneWorld objects in a bidirectional real-time mode. You can use an application to call a OneWorld business function or API and the requested processing is done while you wait. Any return information or feedback is immediately available upon completion of the call.

The following diagram illustrates this model.



The general steps necessary to implement this model follow. These steps should be implemented in the order listed.

Detailed Tasks Using Native APIs

1. Connect to OneWorld.

Connecting to OneWorld establishes a communication link between your application and the OneWorld environment. OneWorld APIs become available for your use and all OneWorld security features are enforced.

2. Call OneWorld business functions.

Once the connection has been established, you may make as many direct calls to OneWorld APIs as is needed to process your transactions. Refer to the Online APIs and the *Development Tools Guide* for more information about OneWorld APIs.

3. Disconnect from OneWorld.

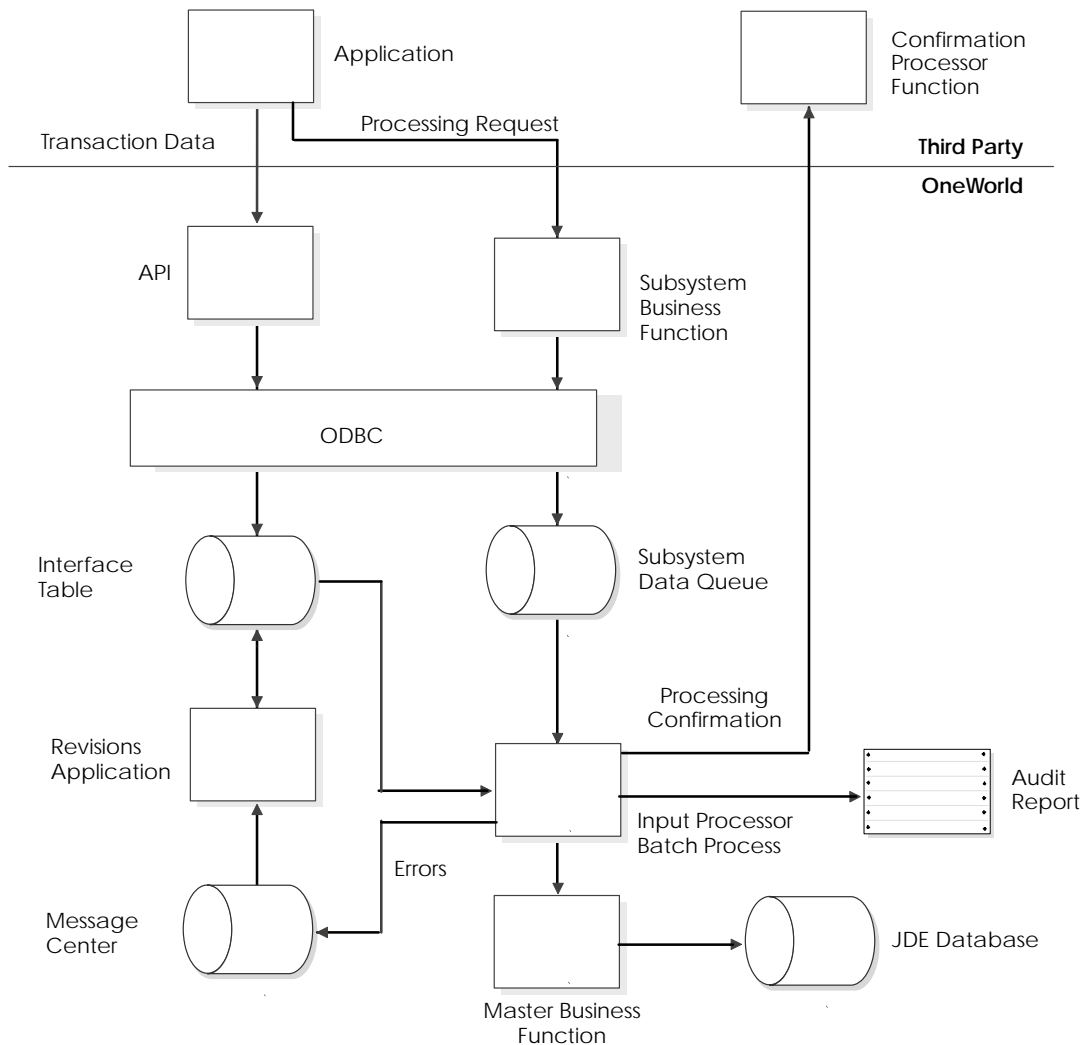
After you make the necessary calls to process your transactions, you must disconnect from OneWorld. This frees resources and prevents further processing with that particular connection.

Some OneWorld APIs may have more parameters than you need. You can create your own simplified interface layer over these APIs.

Implementing Asynchronous Transactions Into OneWorld

Asynchronous transactions into OneWorld allow you to send transactions into OneWorld one at a time without having to wait for completion before you continue processing. You place a transaction in an input table and place a processing request in a queue. A OneWorld background process handles the transaction as it comes up in the queue. When processing is complete, OneWorld calls the function specified in the request to notify you of the status of your process.

The following diagram illustrates this model.



The general steps necessary to implement this model follow. These steps should be implemented in the order listed.

Detailed Tasks Using Native APIs

1. Connect to OneWorld.

Connecting to OneWorld establishes a communication link between your application and the OneWorld environment. OneWorld APIs become available for your use and all OneWorld security features are enforced.

2. Add Records to Interface Tables.

Writing your transaction to the appropriate OneWorld interoperability interface table, makes it available to the OneWorld system for processing.

3. Place an entry in the subsystem data queue.

Place your request for processing in a queue with similar requests. The information you include in the request tells OneWorld the type of transaction and the specific key within the input table so that OneWorld can find it.

You can request confirmation of the inbound transaction by providing the name and library of a custom function to process the confirmation. When processing of the transaction is complete, OneWorld calls the custom function and sends it a flag indicating success or failure.

4. Disconnect from OneWorld.

After you make the necessary calls to process your transactions, you must disconnect from OneWorld. This frees resources and prevents further processing with that particular connection.

5. Process the inbound transaction confirmation.

This step is optional. If you request confirmation, you must write a function that you name, but write to an interface that J.D. Edwards specifies. OneWorld calls this function when the transaction completes. The transaction key from the interface table and a flag that indicates whether the transaction was successful are passed to the function. You include logic in your function to take appropriate action based on the success or failure of the transaction.

If you create a transaction confirmation function, you can also use the function to:

Update your original transaction

By creating a cross-reference between the original transaction and the transaction written to the interoperability table, you can access the original transaction and update it as completed or in an error status.

Using the key returned to this function, you can access the transaction written to the interoperability interface table and retrieve any calculated or defaulted information to update your original transaction.

Run other non-J.D. Edwards business processes

If your transaction is complete, you may want to run a business process that completes the transaction in the non-J.D. Edwards software.

Send messages to users

You may want to inform your users of the status of their original transactions.

Detailed Tasks for OneWorld Operations

1. Run the Subsystem Job from a OneWorld Menu.

In order for the system to respond to transactions one at a time as they come in, you must run a batch process specific to that transaction in subsystem mode. When you place an entry into the subsystem data queue, the batch process starts and processes the transaction.

2. Check for Errors.

When errors occur during inbound processing, they are recorded on the Processor Audit Trail Report and a message is sent to the Employee Work Center. When you review the errors in the work center, you can link directly to the associated transaction in the interface table to make corrections. You can resubmit individual corrected transactions for immediate processing or you can correct all transaction errors and then resubmit them all at once in batch process.

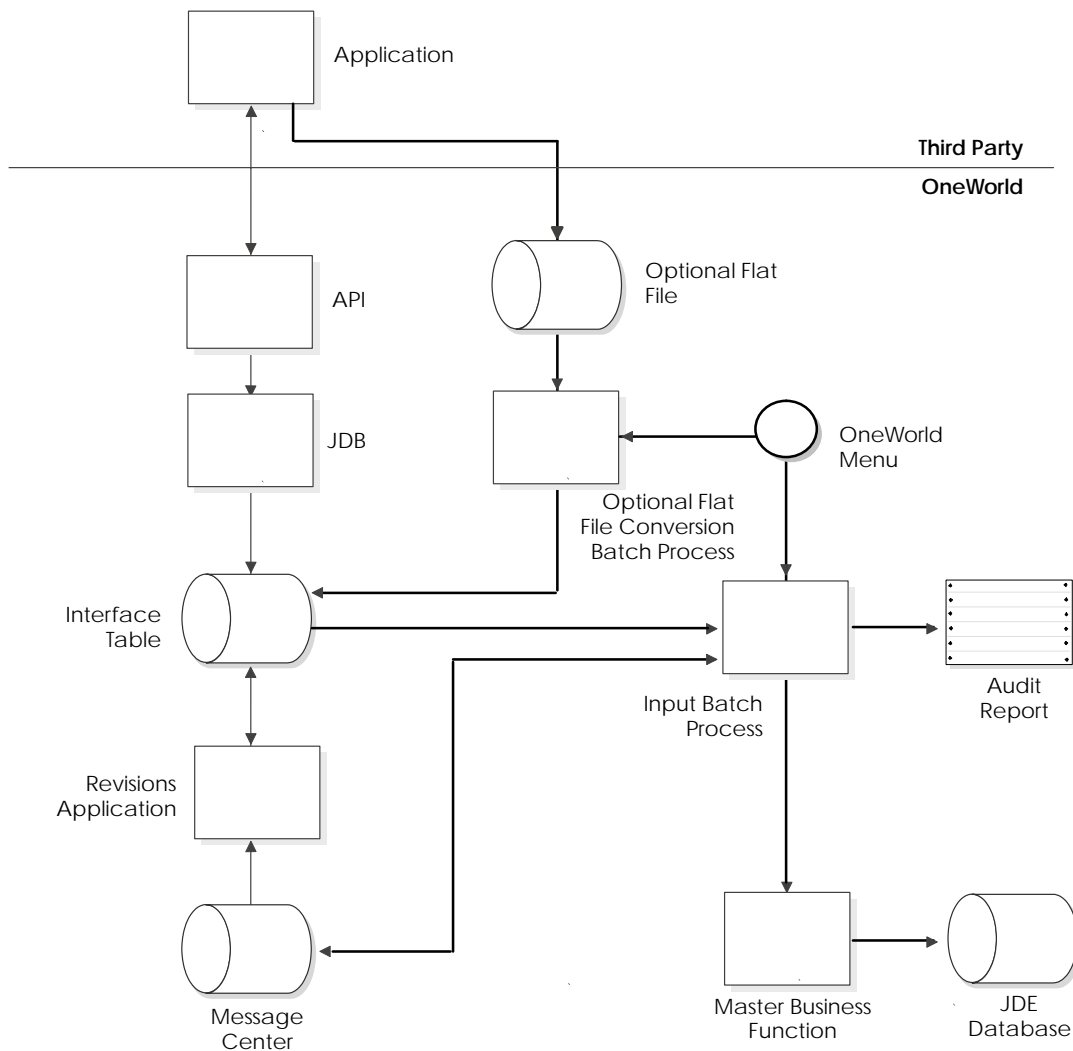
3. Use the Revisions Application.

The revisions application is used to add, delete, edit, and review transactions in the interface tables. For example, if there was an error while processing inbound transactions, you can use this application to correct the record in error.

Implementing Batch Transactions Into OneWorld

Batch transactions into OneWorld allow you to place a large number of transactions into an interface table and process them all at once in batch mode. You initiate the process through the OneWorld menu driver. Errors are placed on an audit report and are also sent to the OneWorld message center. A revisions application allows you to make corrections to the interface table.

The following diagram illustrates this model.



The general steps necessary to implement this model follow. These steps should be implemented in the order listed.

Detailed Tasks Using Native APIs

1. Connect to OneWorld.

Connecting to OneWorld establishes a communication link between your application and the OneWorld environment. OneWorld APIs become available for your use and all OneWorld security features are enforced.

2. Add Records to Interface Tables.

By writing your transaction to the appropriate OneWorld interoperability input table, you make it available to the OneWorld system for processing.

See *Additional Options*.

3. Disconnect from OneWorld.

After you make the necessary calls to process your transactions, you must disconnect from OneWorld. This frees resources and prevents further processing with that particular connection.

Detailed Tasks Using OneWorld Operations

1. Run the batch input processor from a OneWorld menu.

You run the batch input processor to process the batches of transactions you have placed in the interface tables and update the OneWorld transaction tables.

2. Check for Errors.

When errors occur during inbound processing, they are recorded on the Processor Audit Trail Report and a message is sent to the Employee Work Center. When you review the errors in the Work Center, you can link directly to the associated transaction in the interface table to make corrections. You can resubmit individual corrected transactions for immediate processing, or you can correct all transaction errors and then resubmit them all at once in a batch process.

3. Use the Revisions Application.

The revisions application is used to add, delete, edit, and review transactions in the interface tables. For example, if an error occurs while processing inbound transactions, you can use the revision application to correct the record in error.

Additional Options

You can use a comma-delimited flat file for your transactions into OneWorld. To do so, replace steps 1, 2, and 3 in *Detailed Tasks Using Native APIs* above with the following steps:

1. Create a flat file.

Create a flat file in appropriate file format and use a name that correctly matches the transaction type you are using.

2. Run a table conversion.

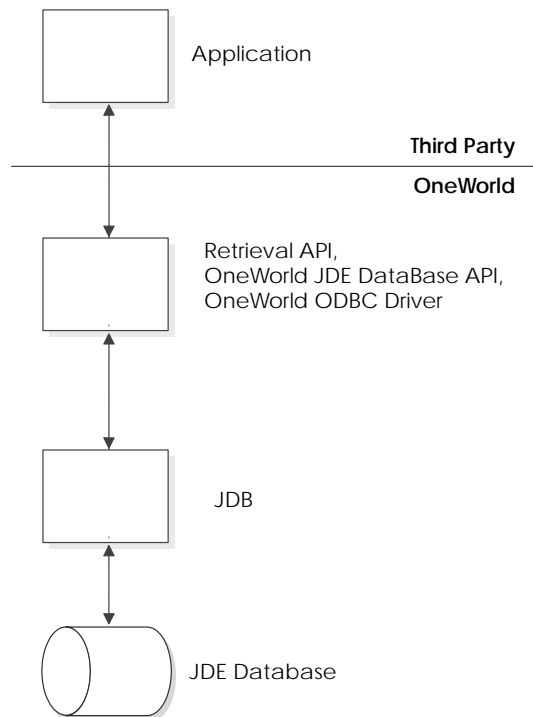
Run a table conversion to convert your flat file into the format that OneWorld expects and then place it in the appropriate interface table.

For more information about using flat files refer to the detailed task, *Importing from Flat Files*, in *Detailed Tasks for OneWorld Operations*.

Implementing Synchronous Transactions From OneWorld

Synchronous transactions from OneWorld allow you to communicate directly with OneWorld objects in a bidirectional real-time mode. You can call a OneWorld business function or API to request information to be returned while you wait. Any return information or feedback is immediately available upon completion of the call.

The following diagram illustrates this model.



The general steps necessary to implement this model follow. These steps should be implemented in the order listed.

Detailed Tasks Using Native APIs

1. Connect to OneWorld.

Connecting to OneWorld establishes a communication link between your application and the OneWorld environment. OneWorld APIs become available for your use and all OneWorld security features are enforced.

2. Call OneWorld Business Functions.

After you are connected to OneWorld, you can make direct calls to OneWorld APIs to retrieve your information. OneWorld retrieval APIs usually do some calculations or interpretation of the database for you. Use the online OneWorld APIs to determine which APIs to use and to obtain any API-specific instructions.

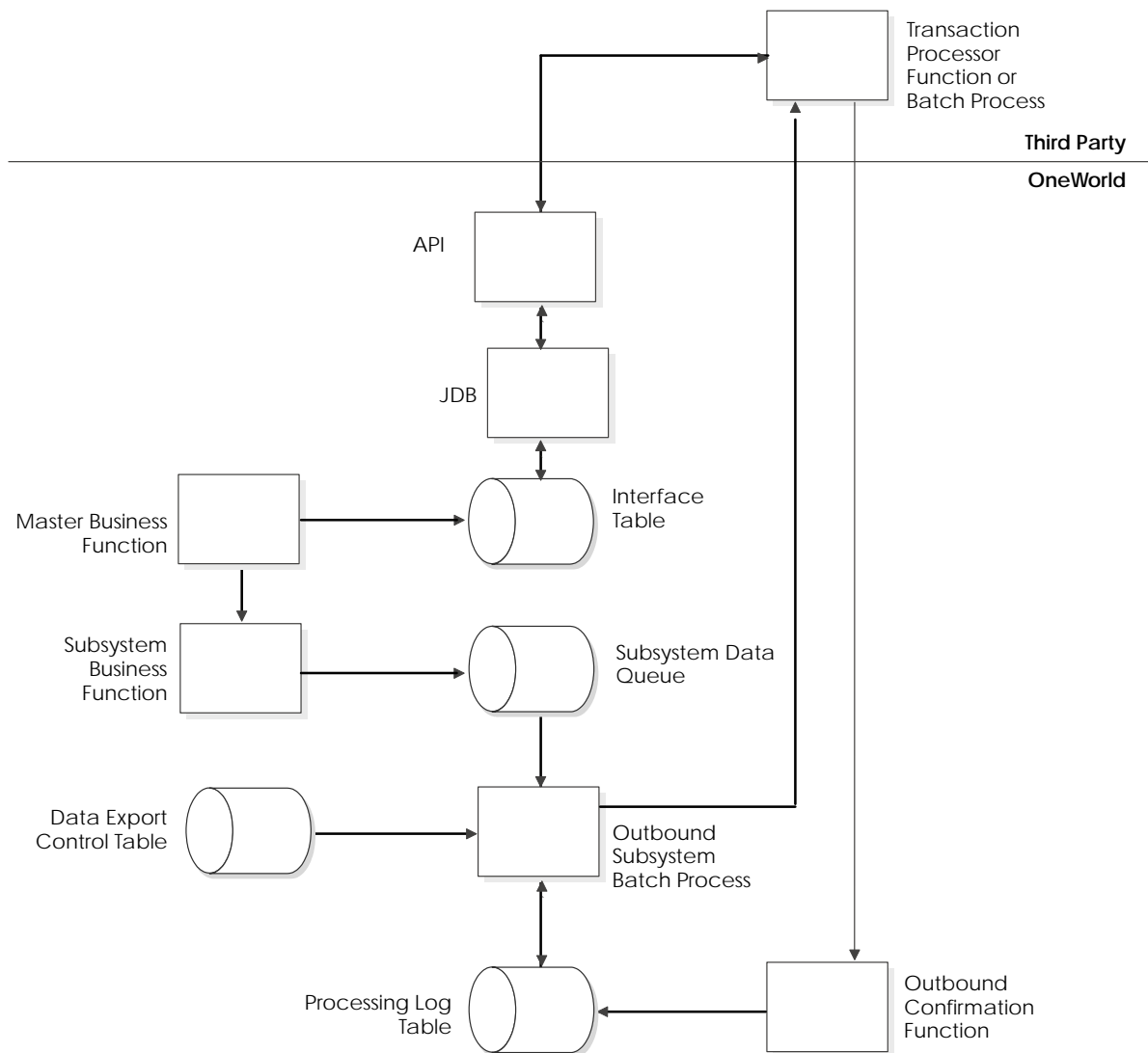
3. Disconnect from OneWorld.

After you make the necessary calls to process your transactions, you must disconnect from OneWorld. This frees resources and prevents further processing with that particular connection.

Implementing Asynchronous Transactions from OneWorld

Asynchronous transactions from OneWorld allow you to receive certain types of OneWorld transactions as they occur. You designate which transactions you want and, when these transactions occur, they are logged to an output table. You are notified individually for each requested transaction when it is available for processing. Each transaction is kept in persistent storage until you notify OneWorld that you are done with it.

The following diagram illustrates this model.



The general steps necessary to implement this model follow. These steps should be implemented in the order listed.

Detailed Tasks Using OneWorld Operations

1. Enable Outbound Transaction Processing.

The creation of outbound transactions is controlled through the processing options associated with the master business functions that control updates to a given table. You must set the appropriate processing option to enable the outbound process for a given transaction type. This process tells the system that the particular transaction type needs to go somewhere, but it does not designate where.

2. Subscribe to the Outbound Transaction.

If you require that a given transaction type is sent to one or more third-party applications, you can associate a given transaction type with each of its individual destinations. You make an entry into the data export control table for each destination. The suggested technique is to specify the name of a third-party function that is called for each transaction as it occurs. Enough information is provided to notify you of the transaction, and give you the key values so you can retrieve it.

3. Run the Subsystem Job from a OneWorld Menu.

This subsystem job creates an entry in the Processing Log tables for each record in the Data Export Control table that subscribes to the specific transaction. It then launches the transaction processor function or batch process, which processes the entry in the interface table. This subsystem job is generic because it does not directly update the transaction specific interface tables.

4. Call OneWorld Completion Confirmation API.

You notify OneWorld when you have completed processing a transaction. OneWorld then marks the processing log entry for that transaction as complete so that the transaction can be purged. The transaction remains in persistent storage so that it can be reprocessed as needed until you purge it. You should use the steps for the Into OneWorld Synchronous methods to call this API.

5. Check for Errors.

When errors occur during processing, they are recorded on the Processor Audit Trail Report and a message is sent to the Employee Work Center. When you review the errors in the Work Center, you can link directly to the associated transaction in the interface table to make corrections. You can resubmit individual corrected transactions for immediate processing or

you can correct all transaction errors and then resubmit them all at once in a batch process.

6. Use the Revisions Application.

Revision applications are used to add, delete, edit, and review transactions in the interface tables. For example, if an error occurs while processing inbound transactions, you can use the revision application to correct the record in error.

Detailed Tasks Using Custom Programming

1. Process the outbound transaction.

This is the function you identified in step 2 of the *Detailed Tasks Using OneWorld Operations*. You can write and deliver this function outside of OneWorld, but it must conform to the interface J.D. Edwards defines. You write this function specifically to handle a given transaction type. You can name it what you want, but it must reside on the same server where the subsystem job is running. The input parameters include the key for the transaction in the OneWorld interface table. Your function then retrieves the transaction and does whatever is necessary with it. You should use the steps for the From OneWorld Synchronous method to do the retrieval.

You can also use this function to:

- Update third-party tables with J.D. Edwards table information.

You use the key for the transaction in the OneWorld interface table to retrieve the transaction and pass the information to a third-party API.

You can also call the OneWorld Completion Confirmation API when processing is finished.

- Store the table information so you can use it for future processing.

You use the key for the transaction in the OneWorld interface table to retrieve the transaction and update a temporary table with the information.

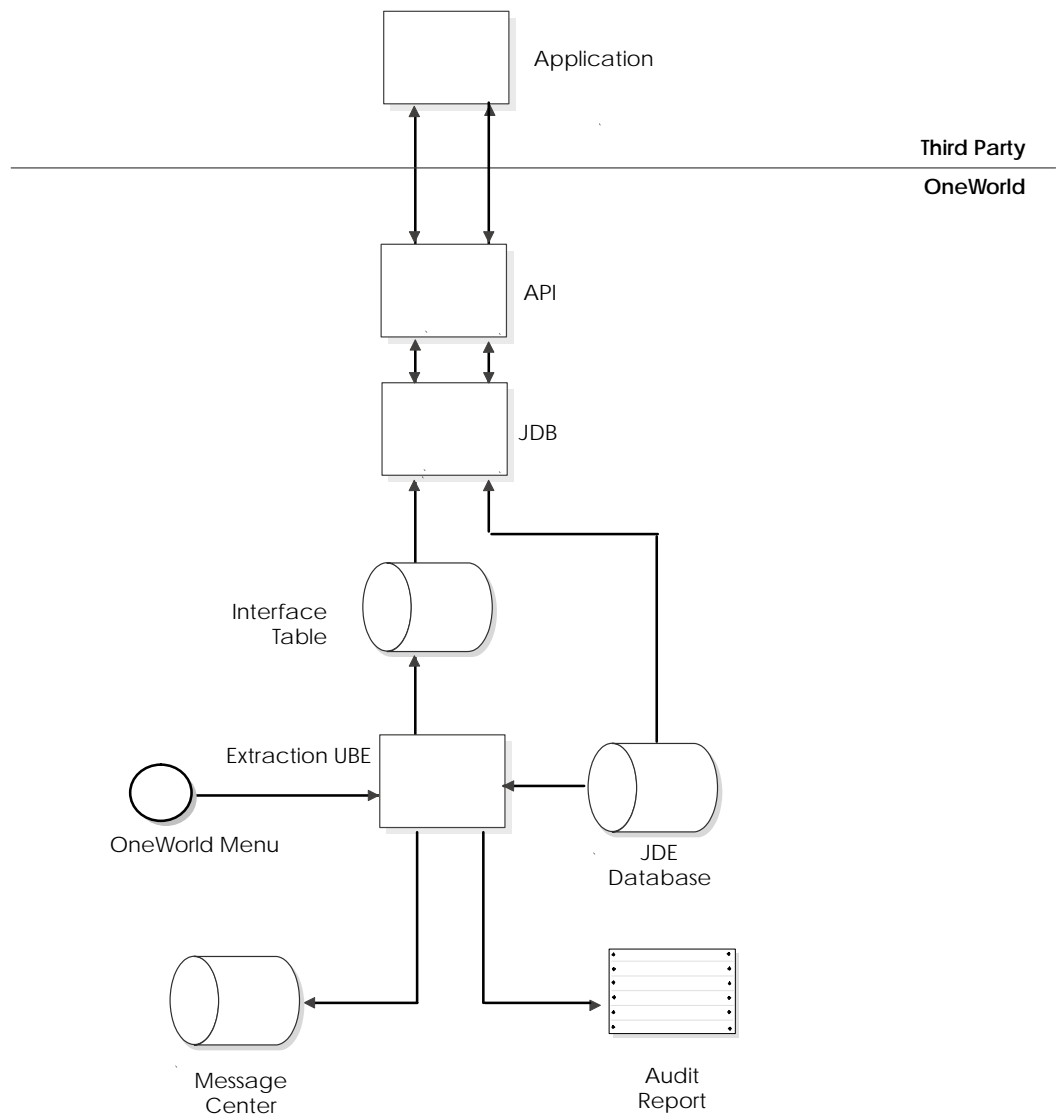
You can then process the data at a later time by using third-party APIs.

You can call the OneWorld Completion Confirmation API when processing is finished.

Implementing Batch Transactions From OneWorld

Batch transactions from OneWorld allow you to place a large number of transactions into an interface table and process them all at once in batch mode. You initiate the process through the OneWorld menu driver. Errors are placed on an audit report and are also sent to the OneWorld message center.

The following diagram illustrates this model.



The general steps necessary to implement this model follow. These steps should be implemented in the order listed.

Detailed Tasks Using OneWorld Operations

1. Run the Extraction Processor from a OneWorld Menu.

The batch extraction processor is used to load the interface tables with data from the permanent OneWorld tables.

Detailed Tasks Using Native APIs

1. Connect to OneWorld.

Connecting to OneWorld establishes a communication link between your application and the OneWorld environment. OneWorld APIs become available for your use and all OneWorld security features are enforced.

2. Retrieve Records from Interface Tables.

The records in the interface table are available for the custom application or batch process to process.

3. Disconnect from OneWorld.

After you make the necessary calls to process your transactions, you must disconnect from OneWorld. This frees resources and prevents further processing with that particular connection.

Additional Options

The model for implementing batch transactions from OneWorld assumes that some form of data manipulation occurs during the batch extraction process. You can extract data directly from permanent OneWorld tables using the same database APIs that you would use to extract data from the interface tables. Instead of opening, retrieving, and closing an interface table, you can perform the same operations on a permanent OneWorld table.



Detailed Tasks for OneWorld APIs

The interoperability tasks you need to perform are based on the method of access you choose to use. You may use any of the following:

- Connect to OneWorld
- Disconnect from OneWorld
- Add records to interface tables
- Retrieve records from interface tables
- Call OneWorld business functions
- Call the OneWorld completion confirmation API
- Place an entry in the subsystem data queue



Connect to OneWorld

The first step in the interoperability process is for two computers to communicate with one another.

Two OneWorld APIs must be called to initiate a OneWorld session:

- JDB_InitEnvOvr
- JDB_InitUser

Both of these APIs must be called before any other OneWorld APIs are called.

When you use these APIs, you may also need to use the following handles:

HENV The environment handle contains information related to the current database connection and valid connection handles. Every application connecting to the database must have an environment handle. This handle is required to connect to a data source.

HUSER The user handle contains information related to a specific connection. Each user handle has an associated environment handle with it. A connection handle is required to connect to a data source.

HREQUEST The request handle contains information related to a specific request to a data source. An application must have a request handle before executing a SQL statement. Each request handle is associated with a user handle.

JDB_InitEnvOvr

The JDB_InitEnvOvr API is used to start an instance of OneWorld. This API should only be called once in the executable. With this API, the JDB_FreeEnv API must also be called at termination of the executable.

Syntax

```
JDEDB_RESULT JDB_InitEnvOvr(HENV *hEnv, char *szEnv, char *szUser, char *szPwd)
```

Parameters

hEnv	Valid handle or NULL. Handle used for other APIs.
szEnv	Name of the environment to initialize.
szUser	OneWorld user name.
szPwd	OneWorld user password.

Return Value

Returns JDEDB_PASSED if the API succeeds. Returns JDEDB_FAILED if the API fails.

JDB_InitUser

The JDB_InitUser API initializes a user in OneWorld. JDB_InitUser is associated with an application rather than a physical user. The user in this instance is an application and is a method of grouping database requests together. Requests are identified with the application that initiated them. Once a user has been registered for an environment handle it cannot be changed. A physical user in OneWorld can initiate multiple applications by calling JDB_InitUser multiple times using a different application name each time.

Syntax

```
JDEDB_RESULT JDB_InitUser(HENV hEnv, HUSER *hUser, char *szApp,  
JDEDB_COMMIT nCommitMode);
```

Parameters

hEnv	Valid environment handle.
hUser	NULL or valid user handle.
szApp	Valid string pointer of active application
nCommitMode	JDEDB_COMMIT_AUTO if transactions are to be committed automatically. JDEDB_COMMIT_MANUAL if transactions are to be committed manually.

Return Value

Returns JDEDB_PASSED if the API succeeds. Returns JDEDB_FAILED if the API fails.

Example

```
/* Declaration of environment variables */
HENV hEnv = NULL;
HUSER hUser = NULL;
JDEDB_RESULT rcode;
char szEnv[11] = "P733ASD1";
char szUser[11] = "UserID";
char szPwd[11] = "Password";

/* Initialize Environment Handle */
if ((rcode = JDB_InitEnvOvr(&hEnv, szEnv, szUser, szPwd)) != JDEDB_PASSED)
{
    printf("JDB_InitEnvOvr failed.\n");
    return (0);
}

/* Initialize User */
if ((rcode = JDB_InitUser(hEnv, &hUser, NULL, JDEDB_COMMIT_AUTO)) != JDEDB_PASSED)
{
    printf("JDB_InitUser failed\n");
    return (0);
}
```


Disconnect from OneWorld

Once you finish your communications between two machines, you must disconnect them.

Two OneWorld APIs must be called to end a OneWorld session:

- JDB_FreeUser
- JDB_FreeEnv

JDB_FreeUser

You use the JDB_FreeUser API to free a user and all related requests. This API must be called for every JDB_InitUser performed.

Syntax

```
JDEDB_RESULT JDB_FreeUser(HUSER hUser);
```

Parameters

hUser	Valid user handle.
-------	--------------------

Return Value

Returns JDEDB_PASSED if the API succeeds. Returns JDEDB_FAILED if the API fails.

JDB_FreeEnv

JDB_FreeEnv is used to free the environment. It frees the environment handle that was initialized by JDB_InitEnv. This function must only be called by executables.

Syntax

```
JDEDB_RESULT JDB_FreeEnv (HENV hEnv);
```

Parameters

hEnv	Valid handle obtained through JDB_InitEnv.
------	--

Return Value

Returns JDEDB_PASSED if the API succeeds. Returns JDEDB_FAILED if the API fails.

Example

```
/* Free user when done processing before freeing the environment */
if (hUser)
{
    if ((rcode = JDB_FreeUser(hUser) != JDEDB_PASSED)
        {
            printf("JDB_FreeUser failed\n");
            return (0);
        }
}

/* Free Environment Handle */
if (hEnv)
{
    if ((rcode = JDB_FreeEnv(hEnv) != JDEDB_PASSED)
        {
            printf("JDB_FreeEnv failed\n");
            return (0);
        }
}
```

Add Records to Interface Tables

Three APIs must be called to insert a record into the transaction-specific interface table:

- JDB_OpenTable
- JDB_InsertTable
- JDB_CloseTable

JDB_OpenTable

JDB_OpenTable opens a table and readies it for input/output processing. This API must be called prior to any other table operations. It initializes the request handle to be associated with each table processing operation.

Syntax

```
JDEDB_RESULT JDB_OpenTable(HUSER hUser, ID idTable, ID idIndex, LPID lpColSelect, unsigned short nNumCols, char * szOverrideDS, HREQUEST * hRequest);
```

Parameters

hUser	Valid user handle.
idTable	Valid table ID of table to be processed.
idIndex	This is the ID of the index to prepare for processing. The index used can be changed after a table has been requested. If zero, the primary index will be used.
lpColSelect	A pointer to an array of column IDs or NULL. These are the columns that will be used. Performance is enhanced if only the required columns are asked for. If all columns are required, then NULL can be passed. The columns must be in sequential order starting from the first column.
nNumCols	Number of columns required or zero. If all columns are required then pass zero.
szOverrideDS	Datasource name. This value overrides the location of the table indicated by the OCM Object Map.
hRequest	NULL or request handle.

Return Value

Returns JDEDB_PASSED if the API succeeds. Returns JDEDB_FAILED if the API fails.

JDB_InsertTable

JDB_InsertTable adds new rows to a specific table.

Syntax

```
JDEDB_RESULT JDB_InsertTable (HREQUEST hRequest, ID idTable, ID idInstance, void * lpStruct);
```

Parameters

hRequest	NULL or request handle.
idTable	Valid table ID of table to be processed.
idInstance	The instance ID of the table. Used if joins, otherwise pass zero.
lpStruct	A pointer to structure or NULL. If no structure is to be used then pass NULL. This structure contains the data to be inserted into the table.

Return Value

Returns JDEDB_PASSED if the API succeeds. Returns JDEDB_FAILED if the API fails.

JDB_CloseTable

JDB_CloseTable releases or frees a request obtained through JDE_OpenTable. A request cannot be used after it has been freed.

Syntax

```
JDEDB_RESULT JDB_CloseTable(HREQUEST hRequest);
```

Parameters

hRequest	Valid request handle.
----------	-----------------------

Return Value

Returns JDEDB_PASSED if the API succeeds. Returns JDEDB_FAILED if the API fails.

Example

This following example contains hard-coded values for illustration purposes.

```

/* Declaration of variables */

HREQUEST hRequest = NULL;
HENV hEnv = NULL;
HUSER hUser = NULL;
NID idTable = NID_F0101Z2;
ID idIndex = ID_F0101Z2_USERID__BATCHNUMBER;
F0101Z2 dsStructToInsert;
JDEDB_RESULT rcode;

/* Open Z table for processing. All columns will be used. */
if ((rcode = JDB_OpenTable(hUser, idTable, idIndex, NULL, (int)0, NULL, &hRequest)) !=
    JDEDB_PASSED)
{
    printf("JDB_OpenTable failed\n");
    return (0);
}

/* Fill in the structure containing data you wish to insert */
strncpy(dsStructToInsert.szedus, "AA1234567", sizeof(dsStructToInsert.szedus));
strncpy(dsStructToInsert.szedbt, "123456789", sizeof(dsStructToInsert.szedbt));
strncpy(dsStructToInsert.szedtn, "000001", sizeof(dsStructToInsert.szedtn));
ParseNumericString(&dsStructToInsert.szedln, "1");
strncpy(dsStructToInsert.sztytn, "JDEAB", sizeof(dsStructToInsert.sztytn));
dsStructToInsert.szdrin = '1';
dsStructToInsert.szedsp = 'N';
strncpy(dsStructToInsert.sztnac, "A", sizeof(dsStructToInsert.sztnac));
strncpy(dsStructToInsert.szalky, "EmployeeName", sizeof(dsStructToInsert.szalky));
strncpy(dsStructToInsert.sztax, "987654321", sizeof(dsStructToInsert.sztax));
strncpy(dsStructToInsert.szat1, "E", sizeof(dsStructToInsert.szat1));
strncpy(dsStructToInsert.szalph, "Employee Name", sizeof(dsStructToInsert.szalph));

/* Insert row into table */

if ((rcode = JDB_InsertTable(hRequest, idTable, 0, &dsStructToInsert)) != JDEDB_PASSED)
{
    printf("JDB_InsertTable failed\n");
    return (0);
}

/* Free the table processing request handle ,hRequest, which was initialized by JDB_OpenTable */
if ((rcode = JDB_CloseTable(hRequest)) != JDEDB_PASSED)
{
    printf("JDB_CloseTable failed\n");
    return (0);
}

```


Retrieve Records from Interface Tables

There are several database APIs that can be used to retrieve records from the transaction-specific interface tables:

- JDB_OpenTable
- JDB_SelectAll
- JDB_Fetch
- JDB_CloseTable

For information regarding the other database APIs, please refer to the online API documentation.

JDB_OpenTable

JDB_OpenTable opens a table and readies it for input/output processing. This API must be called prior to any other table operations. It initializes the request handle associated with each table processing operation.

Syntax

```
JDEDB_RESULT JDB_OpenTable(HUSER hUser, ID idTable, ID idIndex, LPID lpColSelect, unsigned short nNumCols, char * szOverrideDS, HREQUEST * hRequest);
```

Parameters

hUser	Valid user handle.
idTable	Valid table ID of table to process.
idIndex	The ID of the index to ready for processing. The index used can be changed after a table has been requested. If zero, the primary index is used.
lpColSelect	A pointer to an array of column IDs or NULL. These are the columns that will be used. Performance is enhanced if only the required columns are asked for. If all columns are required, then NULL can be passed. The columns must be in sequential order starting from the first column.
nNumCols	Number of columns required or zero. If all columns are required then pass zero.

szOverrideDS	Datasource name. This overrides the location of the table indicated by the Object Map.
hRequest	NULL or request handle.

Return Value

Returns JDEDB_PASSED if the API succeeds. Returns JDEDB_FAILED if the API fails.

JDB_SelectAll

JDB_SelectAll performs a select on a table without a WHERE clause. This causes all records to be selected.

Syntax

```
JDEDB_RESULT JDB_SelectAll(HREQUEST hRequest);
```

Parameters

hRequest	Valid request handle.
----------	-----------------------

Return Value

Returns JDEDB_PASSED if the API succeeds. Returns JDEDB_FAILED if the API fails.

JDB_Fetch

The JDB_Fetch API fetches results of a select. Results are fetched in order.

Syntax

```
JDEDB_RESULT JDB_Fetch(HREQUEST hRequest, void * lpStruct, int nNotUsed);
```

Parameters

hRequest	Valid request handle.
lpStruct	A pointer to results structure or NULL. If no structure is to be used then pass NULL.
nNotUsed	NULL. Currently not in use.

Return Value

Returns JDEDB_PASSED if the API succeeds. Returns JDEDB_FAILED if the API fails.

JDB_CloseTable

JDB_CloseTable frees a request obtained through an JDE_OpenTable. A request cannot be used after it has been freed.

Syntax

```
JDEDB_RESULT JDB_CloseTable(HREQUEST hRequest);
```

Parameters

hRequest	Valid request handle.
----------	-----------------------

Return Value

Returns JDEDB_PASSED if the API succeeds. Returns JDEDB_FAILED if the API fails.

Example

This example contains hard-coded values for illustration purposes.

```
/* Declaration of variables */
HREQUEST hRequest = NULL;
HENV hEnv = NULL;
HUSER hUser = NULL;
NID idTable = NID_F0101Z2;
ID idIndex = ID_F0101Z2_USERID__BATCHNUMBER;
F0101Z2 dsStructToInsert;
JDEDB_RESULT rcode;
int nNotUsed = 0;

/* Open Z table for processing. All columns will be used. */
if ((rcode = JDB_OpenTable(hUser, idTable, idIndex, NULL, (int)0, NULL, &hRequest)) !=
    JDEDB_PASSED)
{
    printf("JDB_OpenTable failed\n");
    return (0);
}

/* Select all of the records from the table */
if ((rcode = JDB_SelectAll(hRequest)) == JDEDB_PASSED)
{
    while((rcode = JDB_Fetch(hRequest, &dsStructToInsert, nNotUsed)) == JDEDB_PASSED)
    {
        /* Process the fetched records */
    }
}
else
{
    printf("JDB_SelectAll failed\n");
    return (0);
}

/* Free the table processing request handle ,hRequest, which was initialized by JDB_OpenTable */
if ((rcode = JDB_CloseTable(hRequest)) != JDEDB_PASSED)
{
    printf("JDB_CloseTable failed\n");
    return (0);
}
```

Call OneWorld Business Functions

OneWorld business functions cannot be called directly from a third-party program. They must be called using the `jdeCallObject` API. Several steps must be taken for `jdeCallObject` to function correctly. This chapter describes how to:

- Initialize the necessary variables
- Call OneWorld business functions using `jdeCallObject`
- Receive errors generated by the business function

For more information regarding OneWorld business functions, see *OneWorld Business Functions in Additional Information* in this guide and *Business Functions* in the *OneWorld Development Tools* guide.

The OneWorld error handling APIs were designed for internal use. They are discussed here to provide you with a means to retrieve application errors while calling master business functions. The structure and functionality of these APIs may change with newer releases of OneWorld. Use these APIs with caution and retest your programs after upgrading to newer releases of OneWorld. The error handling APIs are denoted with double asterisks (**).

Variable Initialization

The `jdeCallObject` API requires two data structures:

- `lpBhvrCom`
- `lpVoid`

<code>lpBhvrCom</code>	A pointer to a data structure used for communication with business functions. Includes an environment handle.
<code>lpVoid</code>	A pointer to a void data structure currently used for error processing. Will be used for security in the future.

Members of these structures must be initialized using the OneWorld `jdeCreateBusinessFunctionParms` API. The memory allocated by these members must be de-allocated at the end of the program. The OneWorld API `jdeFreeBusinessFunctionParms` API performs this task.

jdeCreateBusinessFunctionParms

jdeCreateBusinessFunctionParms initializes members of lpBhvrCom and lpVoid to be used in external calls to business functions.

Syntax

```
BOOL jdeCreateBusinessFunctionParms(HUSER hUser, LPBHVRCOM * lpBhvrCom, LPVOID * lpVoid)
```

Parameters

Parameter	Notes	Usage
hUser	Valid user handle.	Required
lpBhvrCom	Standard lpBhvrCom.	Required
lpVoid	Standard lpVoid.	Required

Return Value

Returns TRUE if the variables were successfully initialized, otherwise returns FALSE.

jdeFreeBusinessFunctionParms

JdeFreeBusinessFunctionParms de-allocates memory used by members of lpBhvrCom and lpVoid that was allocated in the jdeCreateBusinessFunctionParms API.

Syntax

```
void jdeFreeBusinessFunctionParms(lpBhvrCom, lpVoid)
```

Parameters

Parameter	Notes	Usage
lpBhvrCom	Standard lpBhvrCom.	Required
lpVoid	Standard lpVoid.	Required

Return Value

Nothing.

jdeErrorInitializeEx**

jdeErrorInitializeEx allocates memory and initializes an error-related member of lpVoid.

To prepare for error handling, two more members of the lpVoid structure must be initialized. The first member can be initialized using the jdeErrorInitializeEx API. The memory allocated by this API must be de-allocated at the end of the program by the jdeTerminateEx API.

Syntax

```
LPJDEERROR_FORMHDR jdeErrorInitializeEx()
```

Parameters

None.

Return Value

Returns a LPJDEERROR_FORMHDR object, a member of lpVoid.

jdeErrorTerminateEx**

JdeErrorTerminateEx de-allocates memory previously allocated by jdeErrorInitializeEx.

Syntax

```
void jdeErrorTerminateEx (LPJDEERROR_FORMHDR lpHdr)
```

Parameters

Parameter	Notes	Usage
lpHdr	lpHdr initialized in jdeErrorInitializeEx.	Required

Return Value

Nothing.

jdeAlloc

Allocates a variable sized block of memory from a memory pool.

The other member of `lpVoid` needs to be initialized manually, using the `jdeAlloc` API. This memory needs to be de-allocated using the `jdeFree` API.

Syntax

```
Void * jdeAlloc(void * pMemory, unsigned long ulSize, unsigned int nFlags);
```

Parameters

<code>pMemory</code>	Memory pool to allocate from. In most cases will be <code>COM-MON_POOL</code> .	Required
<code>ulSize</code>	Specifies the number of bytes to be allocated.	Required
<code>nFlags</code>	Specifies the characteristics of the allocated memory.	Required

Return Value

If the function succeeds, the return value is a void pointer to the allocated memory. If the function fails, `jdeShutdown` is called to close the application.

Additional Notes

The possible values for the parameter “`nFlags`” are as follows:

<code>MEM_DEFAULT</code>	The allocated memory is not initialized.
<code>MEM_ZEROINIT</code>	The allocated memory is initialized to <code>NULL</code> .
<code>MEM_RESIZEABLE</code>	Space is reserved above the allocated memory to expedite reallocation. Use only when the need to reallocate the memory is anticipated. This can affect performance.

`jdeFree`

The `jdeFree` API frees memory previously allocated from a memory pool.

Syntax

```
unsigned int jdeFree (void *pMemory);
```

Parameters

<code>pMemory</code>	Points to the memory to be freed. (Previously allocated by <code>jdeAlloc</code> or subsequently reallocated by <code>jdeReAlloc</code> .)	Required
----------------------	--	----------

Return Value

If the function succeeds, the return value is non-zero. If the function fails, the return value is zero.

jdeCallObject

You can use JdeCallObject to call master, major, and minor business functions without having to manually build them into the project. Business functions have data structures associated with them, and these must be passed to them through jdeCallObject. The data structures are stored in header files, and must be included in your program. For example, the Address Book MBF Name is N0100041. The header file for this MBF is N0100041.h.

Syntax

```
ID jdeCallObject (char* szFunctionName, LPFNBHVR lpFnBhvr, LPBHVRCOM lpBhvrCom, void* lpVoidInfo, void* lpDS, CALLMAP* lpErrorMap, int nNumMap, char* szLibName, char* szExeLocation, int nFlags);
```

Parameters

Parameter	Notes	Usage
szFunctionName	The literal name of the function placed within double quotes.	Required
lpFnBhvr	Obsolete parameter.	Pass NULL
lpBhvrCom	Standard lpBhvrCom.	Required
lpVoidInfo	Standard lpVoid.	Required
lpDS	Pointer to the data structure of the called business function.	Required
lpErrorMap	Pointer to the call address of the CALLMAP array.	Optional – used for second level messaging.
nnumMap	The #define variable describing the number of indices used in the CALLMAP array.	Optional – used for second level messaging.
szLibName	Future Use.	Required
szExeLocation	Future Use.	Required
nflags	Error/Warning suppress.	Optional

Return Value

Returns 0 for success, 1 for warnings, and 2 for errors.

Application Error Retrieval

When application errors occur during a call to a business function, they are stored in a linked list. OneWorld APIs provide a means for retrieving these errors.

jdeErrorGetCountEx**

Returns the total number of errors and warnings. The second and the third parameters are pointers to the number of errors and warnings.

Syntax

```
int jdeErrorGetCountEx (LPBHVRCOM lpBhvrCom, LPINT lpiErrorCount, LPINT lpiWarningCount, LPVOID lpVoid )
```

Parameters

Parameter	Notes	Usage
lpBhvrCom	Standard lpBhvrCom.	Required
lpiErrorCount	Pointer to number of errors.	Required
lpiWarningCount	Pointer to number of warnings.	Required
lpVoid	Standard lpVoid.	Required

Return Value

Returns the number of errors plus the number of warnings.

jdeErrorSetToFirstEx**

Initializes the error list pointer to the beginning of the list.

Syntax

```
void jdeErrorSetToFirstEx (LPBHVRCOM lpBhvrCom, LPVOID lpVoid)
```

Parameters

Parameter	Notes	Usage
lpBhvrCom	Standard lpBhvrCom.	Required
lpVoid	Standard lpVoid.	Required

Return Value

Nothing.

jdeErrorGetNextDDItemNameInfoEx**

Retrieves the next error in the error list.

Syntax

LPJDEERROR_RECORD jdeErrorGetNextDDItemNameInfoEx (LPBHVRCOM lpBhvrCom, LPVOID lpVoid)

Parameters

Parameter	Notes	Usage
lpBhvrCom	Standard lpBhvrCom.	Required
lpVoid	Standard lpVoid.	Required

Return Value

Returns a JDEERROR_RECORD data structure:

```

struct tagJDEERROR_RECORD
{
    HWND          hwndCtrl;
    int           iGridCol;
    int           iGridRow;
    ERROR_EVENT_KEY zEventKey;
    JDE_ERR_LEVEL iErrorLevel;
    LPSTR         lpszShortDesc;
    NID           szDict;
    int           iNumExtParams;
    LPSTR         *lpIpszParams;
    char          szBFDesc[BLC_MAXLEN_FCNAME + 1];
    LPSTR         lpszSubText;
    char          szFileName[(MAXLEN_PATHNAME +
        MAXLEN_FILENAME + MAXLEN_EXTNAME) * 2];
    int           iLineNo;
    BOOL          bSoundBeep;
    BOOL          bAddToList;
}
    
```

hwndCtrl	Handle to the current window.	Not Applicable
iGridCol	Grid column if error occurred on OneWorld form.	Not Applicable
iGridRow	Grid row if error occurred on OneWorld form.	Not Applicable
zEventKey		Not Applicable
iErrorLevel	1 - Error, 2 - Warning, 3 - Informational Message	
lpszShortDesc	Description of error.	
szDict	Data dictionary ID of the error message.	
iNumExtParams		Not Applicable
*lpIpszParams		Not Applicable
szBFDesc		Not Applicable
lpszSubText		
szFileName	Source file of error message.	
iLineNo	Line number of error message in source file.	
bSoundBeep	True - Sound Beep, False - Do not Sound Beep	
bAddToList	True - Add Error to List, False - Do not Add to List	

Example: Business Function Call

This example illustrates how to use `jdeCallObject` to call the Address Book master business function. This example also illustrates the variable initialization and error handling routines. Values for the master business function data structure have been hard-coded for illustration purposes.

```
#include <n0100041.h>

HUSER          hUser          = NULL;
ID             idResult;
int            j              = 0;
INT            NumErrors      = 0;
INT            NumWarnings    = 0;
LPJDEERROR_RECORD ErrorRec;
DSD0100041     dsAddressBookStruct;

// jdeCallObject variables

LPCG_BHVR      lpVoid         = NULL;
LPVOID         lpDS           = NULL;
LPBHVRCOM      lpBhvrCom      = NULL;
ERROR_EVENT_KEY EventKeyLocal;

// Set up the lpBhvrCom amd lpVoid objects.

jdeCreateBusinessFunctionParms(hUser, &lpBhvrCom,(LPVOID*) &lpVoid);
lpVoid->lpHdr = jdeErrorInitializeEx();
lpVoid->lpErrorEventKey = (LPERROR_EVENT_KEY) jdeAlloc(COMMON_POOL,
    sizeof(ERROR_EVENT_KEY), MEM_ZEROINIT | MEM_FIXED);
lpVoid->lpHdr->nCurDisplayed = -1;
lpBhvrCom->lpObj->lpFormHdr = lpVoid->lpHdr;
EventKeyLocal.hwndCtrl = NULL;
EventKeyLocal.iGridCol = 0;
EventKeyLocal.iGridRow = 0;
EventKeyLocal.wEvent = 1;
lpBhvrCom->lpEventKey = (LPVOID)&EventKeyLocal;
```

```
//Initialize Address Book data structure

memset((void *) &dsAddressBookStruct, (int) '\0', sizeof(DSD0100041));

dsAddressBookStruct.cActionCode = 'A';
dsAddressBookStruct.cUpdateMasterFile = '1';
strncpy(dsAddressBookStruct.szLongAddressNumber, "EmployeeName",
        sizeof(dsAddressBookStruct.szLongAddressNumber));
strncpy(dsAddressBookStruct.szTaxId, "987654321", sizeof(dsAddressBookStruct.szTaxId));
strncpy(dsAddressBookStruct.szSearchType, "", sizeof(dsAddressBookStruct.szSearchType));
strncpy(dsAddressBookStruct.szAlphaName, "Employee Name",
        sizeof(dsAddressBookStruct.szAlphaName));

idResult = jdeCallObject("AddressBookMasterMBF", NULL, lpBhvrCom, lpVoid,
        (void *)&dsAddressBookStruct, (void *)NULL, (int)0, (char *)NULL, (char *)NULL, (int)0);

// Does the call have errors?

if (jdeErrorGetCountEx(lpBhvrCom, &NumErrors, &NumWarnings, lpVoid) > 0)
{
    jdeErrorSetToFirstEx(lpBhvrCom, lpVoid);
    while (ErrorRec = jdeErrorGetNextDDItemNameInfoEx(lpBhvrCom, lpVoid))
    {
        // Process Errors here
    }
}

// Free memory used by lpBhvrCom and lpVoid

jdeFree(((LPCG_BHVR)lpVoid)->lpErrorEventKey);
jdeErrorTerminateEx(((LPCG_BHVR)lpVoid)->lpHdr);
jdeFreeBusinessFunctionParms(lpBhvrCom, lpVoid);
```


Call the OneWorld Completion Confirmation API

Vendor-specific batch processes and functions in the Asynchronous Model call the OneWorld Completion Confirmation API each time they successfully process a record in the interface table. If the vendor-specific batch process or function failed in some way, it does not call this function. When this API is called, the associated record in the Processing Log Table is updated with a 'Y' in the Successfully Processed (EDSP) field.

When you call the OneWorld Completion Confirmation API from a batch process, call it in the same way you would call a business function. The example at the end of this chapter illustrates how to call this API from a third-party function.

InteropOutboundConfirmationFunc

InteropOutboundConfirmationFunc updates the corresponding record in the Processing Log Table with a 'Y' in the Successfully Processed (EDSP) field.

Syntax

```
ID InteropOutboundConfirmationFunc (LPBHVRCOM lpBhvrCom, LPVOID lpVoid, LPDSD0000195B lpDS);
```

Parameters

lpBhvrCom	Standard lpBhvrCom.
lpVoid	Standard lpVoid.
lpDS	Pointer to the data structure of this API.

Return Value

Returns ER_SUCCESS if the API succeeds. Returns ER_ERROR if the API fails.

Data Structure

The data structure for this API is as follows:

Data Item	Description	Required	I/O
EDUS	User ID - 11 Characters	Y	I
EDBT	Batch Number - 16 Characters	Y	I

EDTN	Transaction Number - 23 Characters	Y	I
EDLN	Line Number - Double	Y	I
TYTN	Transaction Type - 9 Characters	Y	I
DCTO	Document Type - 3 Characters	Y	I
SEQ	Sequence Number - Double	N	I
ERR1	Error Encountered - 1 Character	N	O

Example: Vendor Function

This example contains hard-coded values for illustration purposes.

```

/* Declaration of variables */

LPBHVRCOM lpBhvrCom = NULL;
LPVOID lpVoid = NULL;
ID rcode = ER_SUCCESS;
DSD0000195B dsOutboundConfStruct;

/* Fill in the structure containing data you wish to insert */
strncpy(dsOutboundConfStruct.szUserId, "AA1234567", sizeof(dsOutboundConfStruct.szUserId));
strncpy(dsOutboundConfStruct.szBatchNumber, "123456789",
        sizeof(dsOutboundConfStruct.szBatchNumber));
strncpy(dsOutboundConfStruct.szTransactionNumber, "000001",
        sizeof(dsOutboundConfStruct.szTransactionNumber));
ParseNumericString(&dsOutboundConfStruct.dLineNumber, "1");
strncpy(dsOutboundConfStruct.szTransactionType, "JDEAB",
        sizeof(dsOutboundConfStruct.szTransactionType));
strncpy(dsOutboundConfStruct.szOrderType, " ", sizeof(dsOutboundConfStruct.szOrderType));

/* Call Confirmation API */

if ((rcode = InteropOutboundConfirmationFunc (lpBhvrCom, lpVoid, lpDS) != ER_SUCCESS)
    {
    printf("Outbound Confirmation API call failed\n");
    return (0);
    }

```

Place an Entry in the Subsystem Data Queue

Subsystem jobs are continuous jobs that process records from a data queue and run until you request an end to the job. Subsystem jobs read records one at a time for a subsystem table, retrieve information from the particular record, and run a configurable processing engine for each record. At the end of the records, instead of ending the job, subsystem jobs wait for a specific period and then retrieve the information for each record once again. For each subsystem job, there can be multiple records in the subsystem table.

You start a subsystem job as you would a regular batch job. Unlike batch jobs, subsystem jobs can only run on a server.

Before processing, OneWorld makes sure that limits for the subsystem job on the particular server have not been exceeded. If limits have been exceeded, the configurable processing engine will not process the subsystem job.

To add a record to the subsystem table, you use the `jdeCallObject` API. For more information regarding this API, see *Calling OneWorld Business Functions* in *Detailed Tasks for OneWorld APIs* in this guide.

See Also

- The *System Administration Guide* for more information about subsystem jobs

Inbound Transaction Subsystem Data Structure

When you use `jdeCallObject`, you call the `AddInboundTransToSubsystemQueue` business function. This business function is a generic function that writes a record to the subsystem data queue to specify a batch process that must be activated in the subsystem. The data structure to be used in `jdeCallObject` is included in the header file `b0000176.h`.

```
typedef struct tagDSD0000176A
{
    char          szUBENAME[11];
    char          szVersion[11];
    char          szUserId[11];
    char          szBatchNumber[16];
    char          szTransactionNumber[23];
    MATH_NUMERIC mnLineNumber;
    Char          cSuppressErrorMessage;
    char          szFunctionName[51];
    char          szFunctionLibrary[257];
} DSD0000176A, *LPDSD0000176A;
```

SzUBENAME	Subsystem UBE Name.
SzVersion	Subsystem UBE Version.
SzUserId	User ID.
SzBatchNumber	Batch Number.
SzTransactionNumber	Transaction Number.
MnLineNumber	Line Number.
C SuppressErrorMessage	Suppress Error Messages Flag.
SzFunctionName	Confirmation Function Name (Optional).
SzFunctionLibrary	Confirmation Function Library (Optional).

The batch number, transaction number, and line number are the values from the record that were entered into the Interface Table.

Requesting Inbound Transaction Confirmation

You can request an inbound transaction confirmation when records are placed in the subsystem data queue. This confirmation notifies the third party if the inbound transaction is successful. A custom function must be provided to process the confirmation. To request the inbound confirmation, populate the subsystem data structure members SzFunctionName and SzFunctionLibrary with the name of the custom function and the full name of the library containing the function, including the path. For more information about the Confirmation Function, see *Processing the Inbound Transaction Confirmation*.

Example

This example contains hard-coded values for illustration purposes.

```
#include <b0000176.h>

DSD0000176A dsSubsystemStruct;

memset((void *)&dsSubsystemStruct, (int)('\0'), sizeof(DSD0000176A));

strncpy(dsSubsystemStruct.szUBENAME, (const char *) "R01010Z",
        sizeof(dsSubsystemStruct.szUBENAME));
strncpy(dsSubsystemStruct.szVersion, (const char *) "ZJDE0002", sizeof(dsSubsystemStruct.szVersion));
dsSubsystemStruct.cSuppressErrorMessage = '0';
strncpy(dsSubsystemStruct.szUserId, (const char *) "AA1234567", sizeof(dsSubsystemStruct.szUserId));
strncpy(dsSubsystemStruct.szBatchNumber, (const char *) "123456789",
        sizeof(dsSubsystemStruct.szBatchNumber));
strncpy(dsSubsystemStruct.szTransactionNumber, (const char *) "000001",
        sizeof(dsSubsystemStruct.szTransactionNumber));
ParseNumericString(&dsSubsystemStruct.mnLineNumber, "1");
strncpy(dsSubsystemStruct.szFunctionName, (const char *) "InboundConfirm",
        sizeof(dsSubsystemStruct.szFunctionName));
strncpy(dsSubsystemStruct.szFunctionLibrary, (const char *) "confirm.dll",
        sizeof(dsSubsystemStruct.szFunctionLibrary));

rcode = jdeCallObject("AddInboundTransToSubsystemQueue", NULL, lpBhvrCom, lpVoid,
                    (LPVOID) &dsSubsystemStruct, (CALLMAP *)NULL, (int)0, (char *)NULL, (char *)NULL,
                    (int)0);

if (rcode == 0)
{
    printf("Success\n");
}

if (rcode == 1)
{
    printf("Warning\n");
}

if (rcode == 2)
{
    printf("Error\n");
}
```



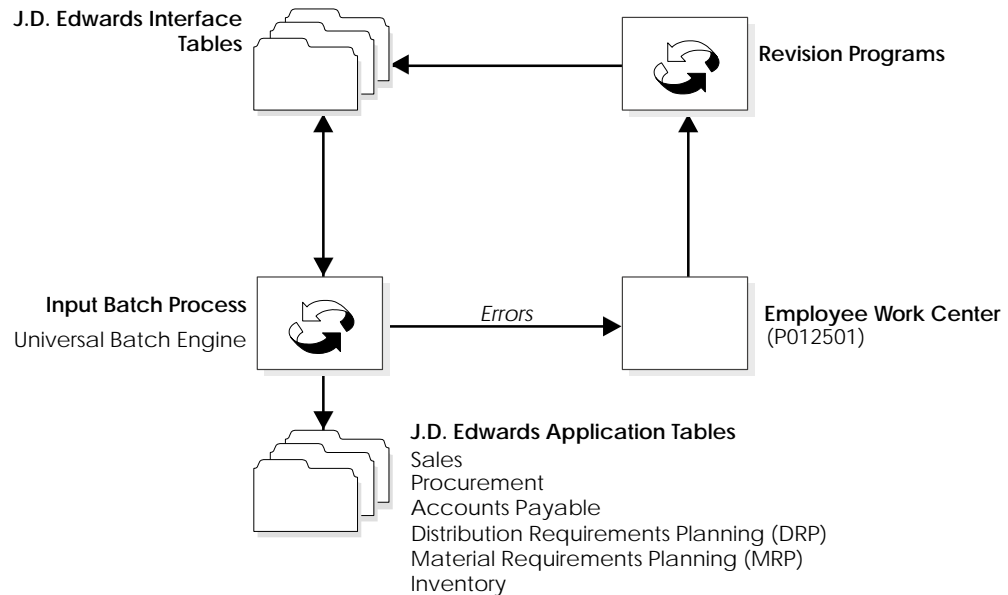

Detailed Tasks for OneWorld Operations

The interoperability tasks you need to perform are based on the method of access you choose to use. The following tasks are the standard OneWorld setup or operational procedures that you may use:

- Run the Input Batch Process
- Run the Extraction Batch Process
- Run a subsystem job from a OneWorld menu
- Enable outbound transaction processing
- Subscribe to outbound transaction
- Check for errors
- Use the Revisions Application
- Import from flat files



Run the Input Batch Process



To update J.D. Edwards applications, run the Input Batch Process. The Input Batch Process Access report contains information specific to each transaction type supported by J.D. Edwards.

The Input Batch Process uses the data in the interface tables to update the appropriate J.D. Edwards application tables. For example, when you receive a purchase order from a third-party application, the Input Batch Process for customer orders updates the sales application and creates a sales order based on the data from the purchase order.

When you choose Input Batch Process, the program displays a version list of report features. You can use an existing report version, change an existing report version, or add a report version. When using a report version, you can change the processing options and data selection to fit your needs.

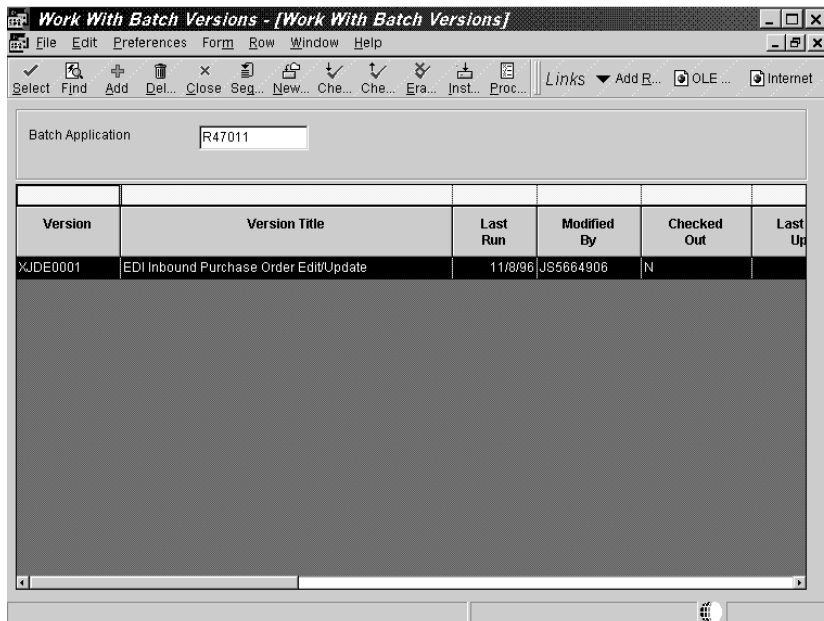
If the Input Batch Process encounters errors while moving the data from the interface tables to the application programs, it sends error messages to the Employee Work Center on the Workflow Management menu (G02).

When the Input Batch Process program finishes, it generates an audit report that lists the transactions that were processed, totals for the number of processed transactions, and errors that occurred during processing.

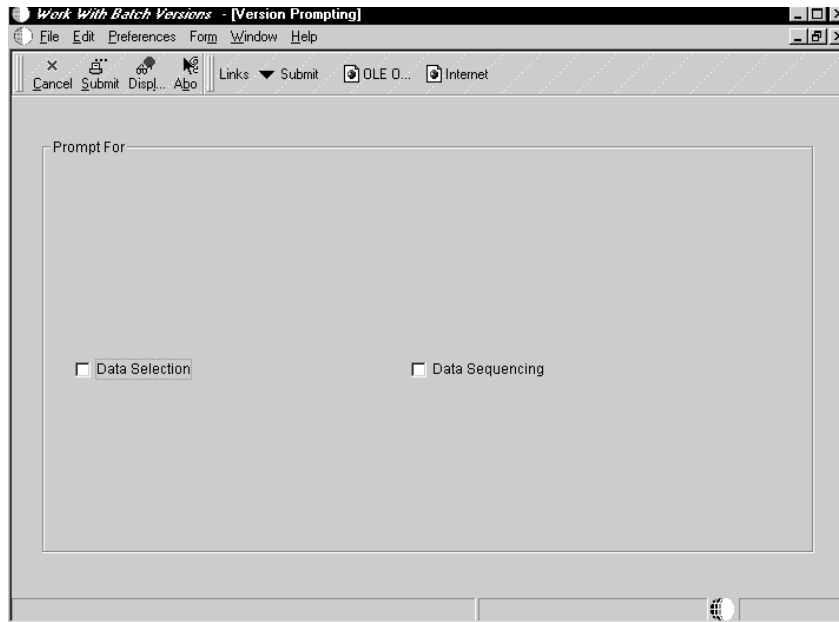
► To run the Input Batch Process

Start the Input Batch Process for the Import transaction type. See *Appendix B* for a list of input batch processes.

1. On Work With Batch Versions, review the processing options to determine how the data is processed by the Inbound Edit/Update program.



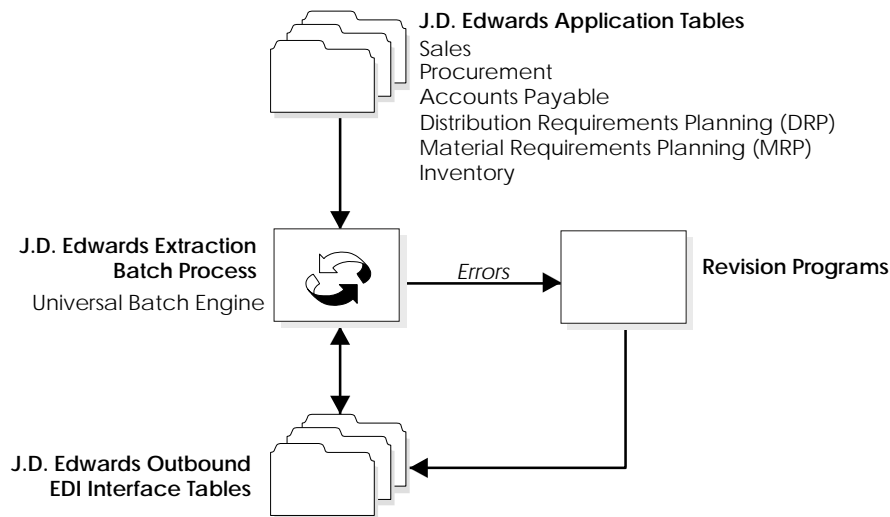
2. Choose the version you want to run and click Select.



3. On Version Prompting, click any of the following to review the report feature options:
 - Data Selection
 - Data Sequencing
4. Click the Submit button.

Field	Explanation
Data Selection	Check box to change data selection before report is submitted or previewed.
Data Sequencing	Check box to change data sequencing before report is submitted or previewed.

Run an Extraction Batch Process



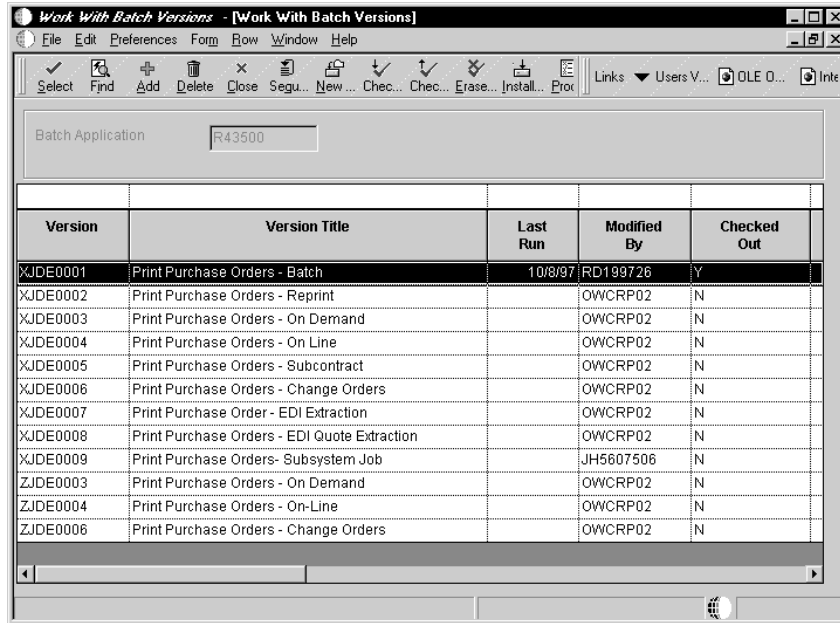
You must copy the records from your J.D. Edwards application tables to the J.D. Edwards outbound interface tables. To copy the records, use the Extraction Batch Process that is specifically set up for the type of document you are sending.

The Extraction Batch Process displays a version list of report features. You can run an existing version, change an existing version, or add a version. You can also change the processing options and data selection options for that version to fit your needs.

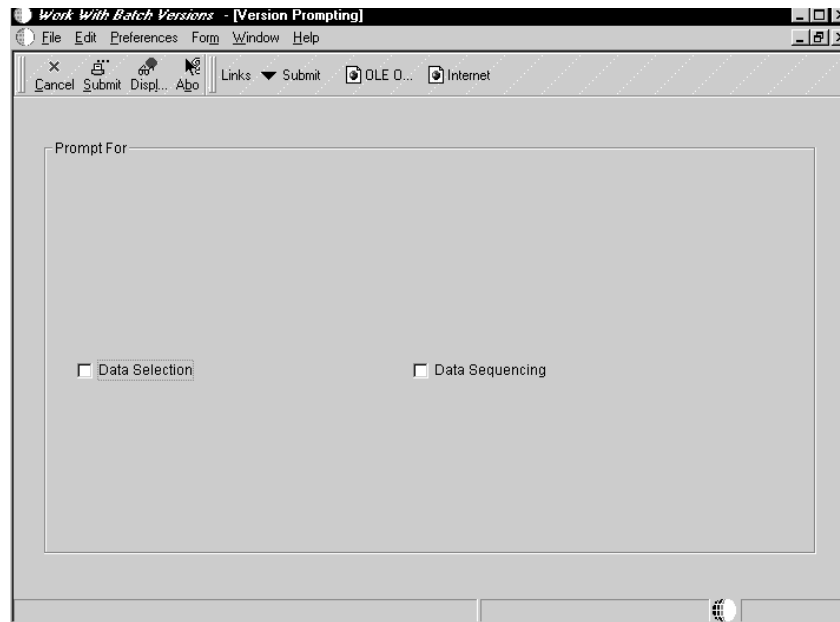
When you run the Extraction Batch Process, the program retrieves data from the J.D. Edwards application tables for the transaction and copies the data into the interface tables. The system also generates an audit report that lists which documents were processed.

► To run the Extraction Batch Process

Start the Extraction Batch Process for the transaction type you want to export. See *Appendix B* for a list of Extraction Batch Processes.



1. On Work With Batch Versions, choose the version you want to run and click Select.



2. On Version Prompting, click any of the following to review the report feature options:
 - Data Selection
 - Data Sequencing
3. Click the Submit button.

The Extraction Batch Process retrieves data from the J.D. Edwards application tables and copies the data into the outbound interface tables.

The Extraction Batch Process program also generates an audit report that lists which documents completed successfully.

Field	Explanation
Data Selection	Check box to change data selection before report is submitted or previewed.
Data Sequencing	Check box to change data sequencing before report is submitted or previewed.

Run a Subsystem Job from a OneWorld Menu

Subsystem jobs are continuous jobs that process records from a data queue. This type of job runs until you request an end to the job. Subsystem jobs read records one at a time for a subsystem table, retrieve information from the particular record, and run a configurable processing engine for each record. At the end of the records, instead of ending the job, subsystem jobs wait for a specific period and then retrieve the information for each record once again. For each subsystem job, there can be multiple records in the subsystem table.

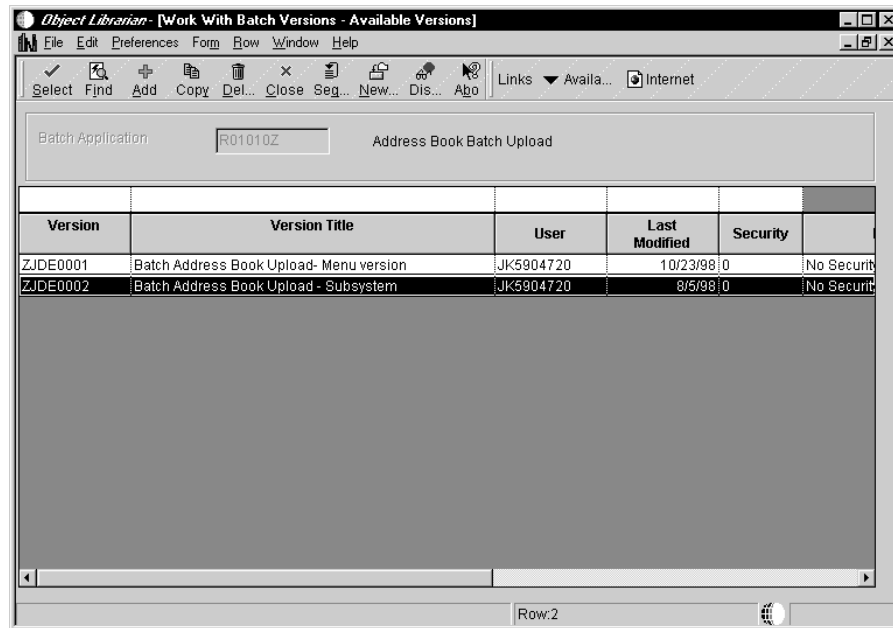
You start a subsystem job as you would a regular batch job. There is no difference between running a subsystem job and running a batch job.

Before processing, OneWorld makes sure that limits for the subsystem job on the particular server have not been exceeded. If these limits are exceeded, the configurable processing engine will not process the subsystem job.

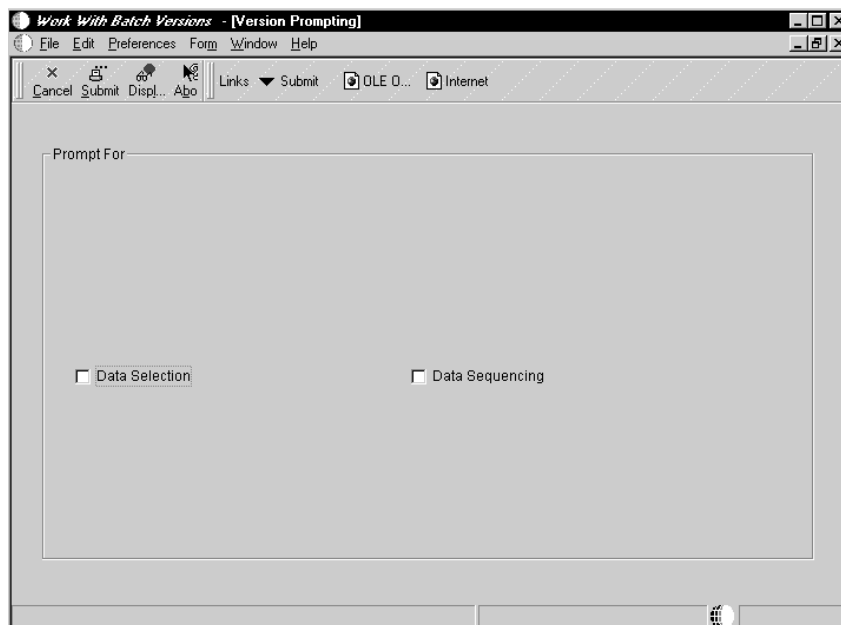
► **To run a subsystem job**

Start either the Inbound Processor batch process for the transaction type you wish to import or the Interoperability Generic Outbound Subsystem batch process (R00460). See *Appendix B* for a list of input process batch processes.

Review the processing options to determine how the data will be processed by the subsystem job.



1. On Work With Batch Versions - Available Versions, choose the subsystem version you want to run and click Select.



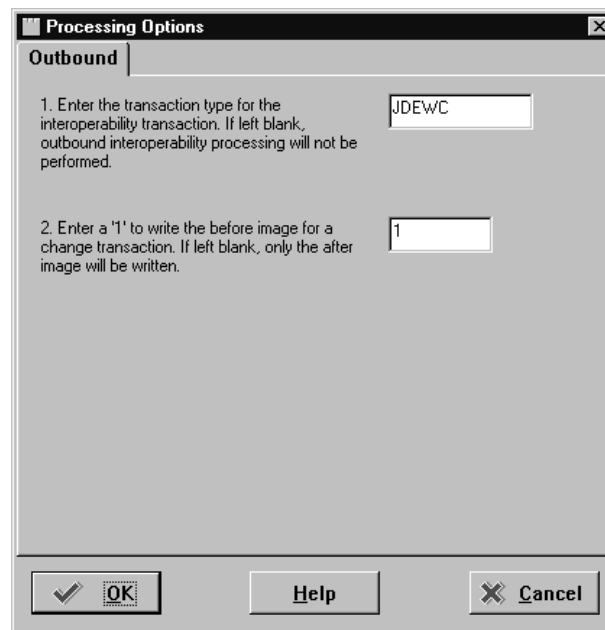
2. On Version Prompting, click any of the following to review the report feature options:
 - Data Selection
 - Data Sequencing
3. Click the Submit button.

Enable Outbound Transaction Processing

All outbound master business functions used to create interoperability transactions have processing options that control how the transaction is written. The first processing option is the transaction type for the interoperability transaction. If this processing option is left blank, outbound interoperability processing will not be performed. The second processing option controls whether the before image is written for a change transaction. If set to 1, the before and after images of the transaction are written to the interface table. If this is not set, then only an after image is written.

► To enable outbound transaction processing

1. Right-click the application that contains the processing options for the transaction's master business function. See *Appendix B* for a list of these applications.
2. Choose Prompt for Values from the menu.



The screenshot shows a dialog box titled "Processing Options" with a tab labeled "Outbound". The dialog contains two text input fields. The first field is labeled "1. Enter the transaction type for the interoperability transaction. If left blank, outbound interoperability processing will not be performed." and contains the text "JDEWC". The second field is labeled "2. Enter a '1' to write the before image for a change transaction. If left blank, only the after image will be written." and contains the text "1". At the bottom of the dialog, there are three buttons: "OK" (with a checkmark icon), "Help", and "Cancel" (with an 'X' icon).

3. Click either the Outbound tab or the Interop tab.
4. Enter the transaction type and set the before image flag if you want before and after images written to the interface table for change transactions, and then click OK.

Subscribe to Outbound Transactions

Subscribing to outbound transactions involves using the following:

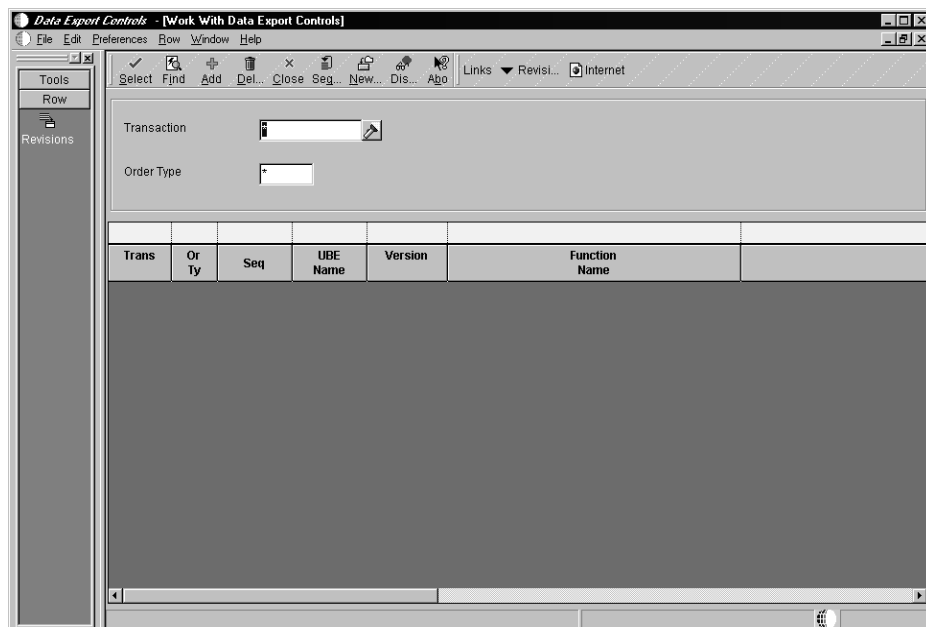
- Data Export Control Table
- Processing Log Table

Using Data Export Controls

The flow of outbound data to third parties is controlled through the Data Export Controls application. For each transaction type and order type, one or more records can be defined with different batch process names and versions, or function names and libraries. This allows for the subscription of multiple vendor-specific objects for an interoperability transaction. Records in this table determine the vendor-specific objects to call from the Outbound Subsystem batch process (R00460) or Outbound Scheduler batch process (R00461) for each transaction processed by the batch process.

► To use Data Export Controls

1. On Work With Data Export Controls, click Add.



2. Complete the following fields:
 - Transaction Type
 - Order Type
3. For each detail row, enter the following:
 - Either a batch process Name and Version, or a Function Name and Library.

Only one set can be entered.

 - Enter 1 in the Execute for Add column if you want the vendor-specific object to be launched for an add/insert.

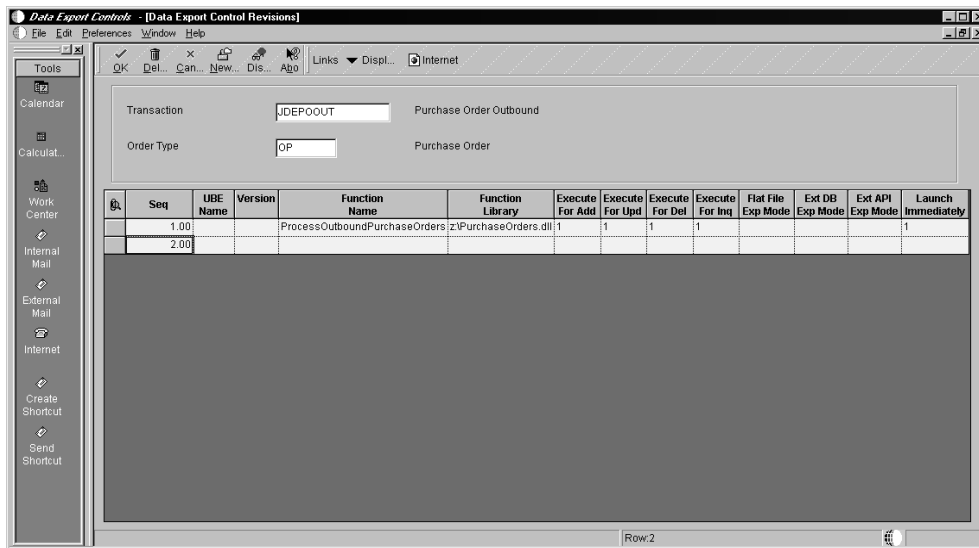
Make the same decision for update, delete, and inquiry and enter 1 in the appropriate column.

 - Enter 1 in the Launch Immediately column to launch the object from the Outbound Subsystem batch process.

This column does not affect the Outbound Scheduler batch process.

In B733, the Export Mode columns are not used.

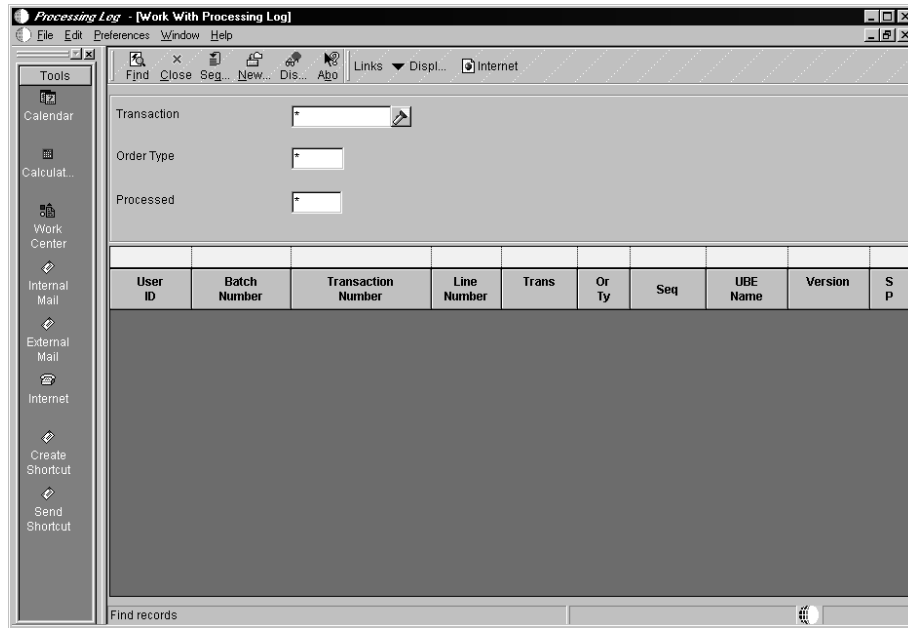
The Sequence Number automatically increments for each line.



Processing Log Table

The Processing Log table contains the keys to the interoperability transaction; the transaction type, order type, and sequence number from the Data Export Control table; the batch process and version from the Data Export Control table if these columns were filled in; and a successfully processed column. A vendor-specific record is sequentially created in this table for every transaction processed by the Outbound Subsystem batch process. For example, if three vendors have subscribed to a transaction using Data Export Controls, three records are created in the Processing Log table for each transaction. If the vendor-specific object successfully processed the transaction, the Processing Log record is updated with a 'Y' in the successfully processed column. You use the Processing Log application (P0046) to determine whether a vendor-specific object processed the interoperability transaction correctly.

Data in the Processing Log table cannot be changed from this application. There is no way to change data in this table through OneWorld applications.

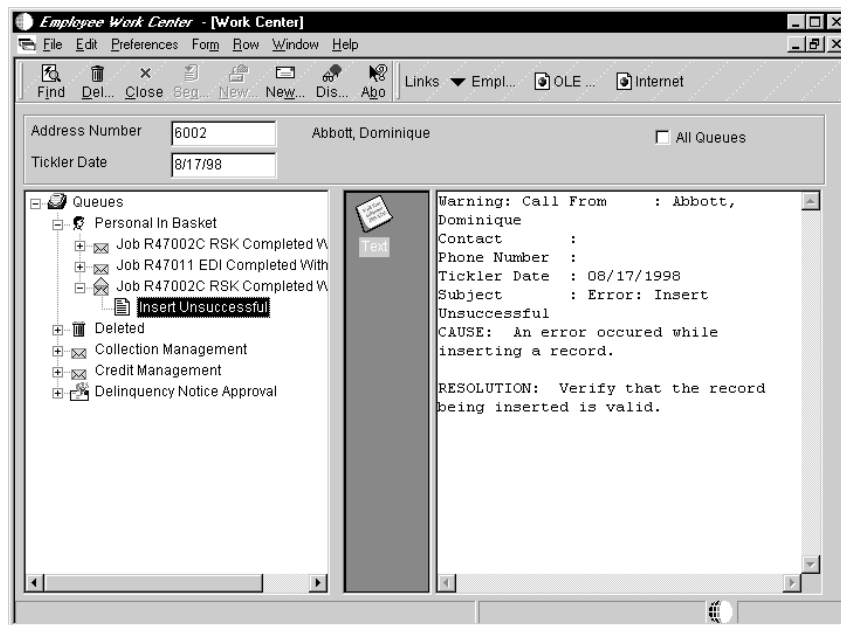


Check for Errors

You can check for errors that occur when running Inbound, Extraction, and Flat File Processing.

Inbound Flat File Processing

If the flat file was not successfully processed, check the Employee Work Center and Status Inquiry programs to determine if any errors have occurred when running the Flat File Conversion program (R47002C). See the *OneWorld Foundation Guide* for the process on checking the Employee Work Center for errors.



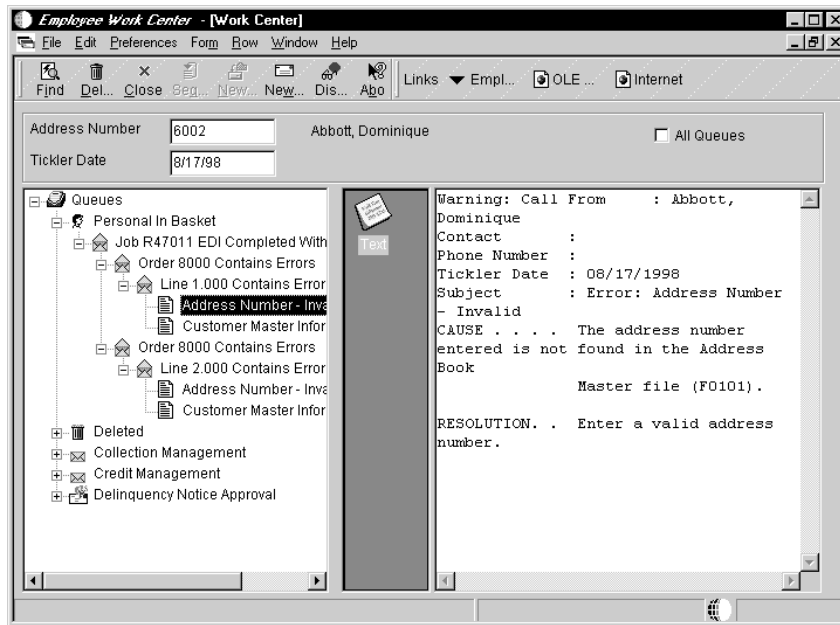
In the example above, an “Insert Unsuccessful” error message was generated due to a record already existing in the interface files with duplicate keys.

If the flat file was not successfully processed, correct the error condition and run the Flat File Conversion program again.

Inbound and Extraction Batch Processing

To determine if an error occurred while running an inbound or extraction batch process, review the Batch Processor Audit Report.

If a “Y” appears in the left-hand column of the report under the Line Contains Errors Y/N heading, an error condition occurred during processing. Access the Employee Work Center for detailed error messages.



In the example above, an error condition occurred while running the Inbound Processor due to the fact that the customer number being processed did not exist in the address book. Ensure that each Sold-To-and Ship-To address is properly set up in the address book and in the Customer Master.

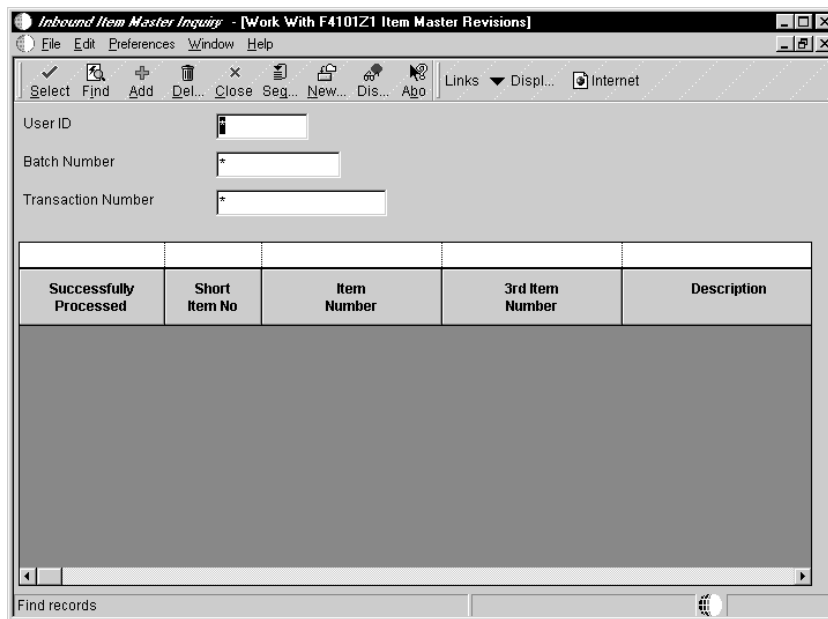
Some error messages might contain a shortcut, which appears as a OneWorld icon in the middle column of the Employee Work Center. When you click this shortcut, the system invokes the Revisions Application, where you can view the record in error and make any necessary changes to that record. After correcting all of the records, you can resubmit the batch processor to process the remaining records.

Use a Revisions Application

Revisions applications are used to add, delete, edit, and review transactions in the interface tables. Use the revision application that is appropriate for the type of transaction you are using. You can use a revisions application to correct the record in error, if there was an error while processing inbound transactions. You then run the transaction process again, continuing to make corrections and rerun the transaction process until the program completes without errors. When deleting records, a revisions application calls the appropriate purge named event rule to delete records. See *Appendix B* in this guide for a list of the revisions applications.

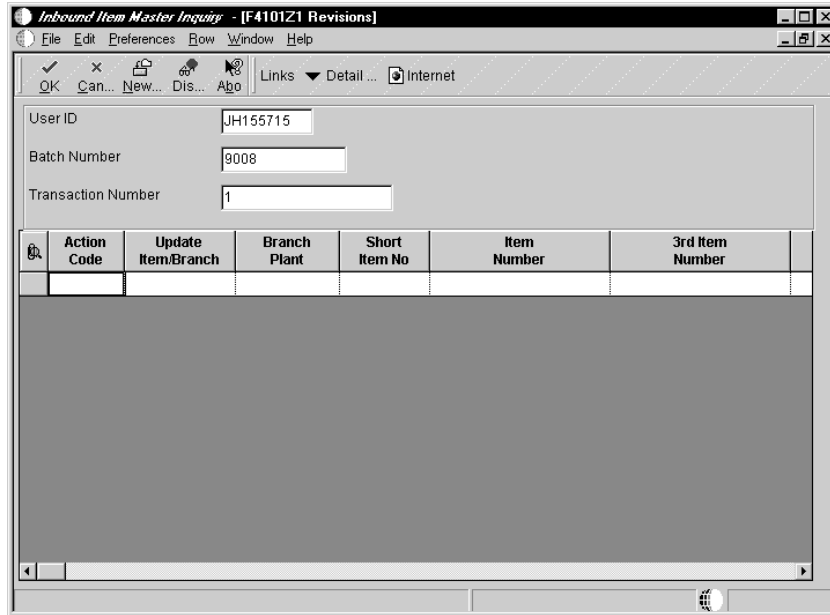
► To review and revise inbound transactions

From the Interoperability menu, choose the Revisions Application you wish to use.



1. On the Revisions Application you wish to use (for example, Work With Item Master Revisions) to limit the search to specific transactions, complete the following fields:
 - User ID
 - Batch Number

- Transaction Number
2. Click Find.
 3. Choose the transaction and click Select.

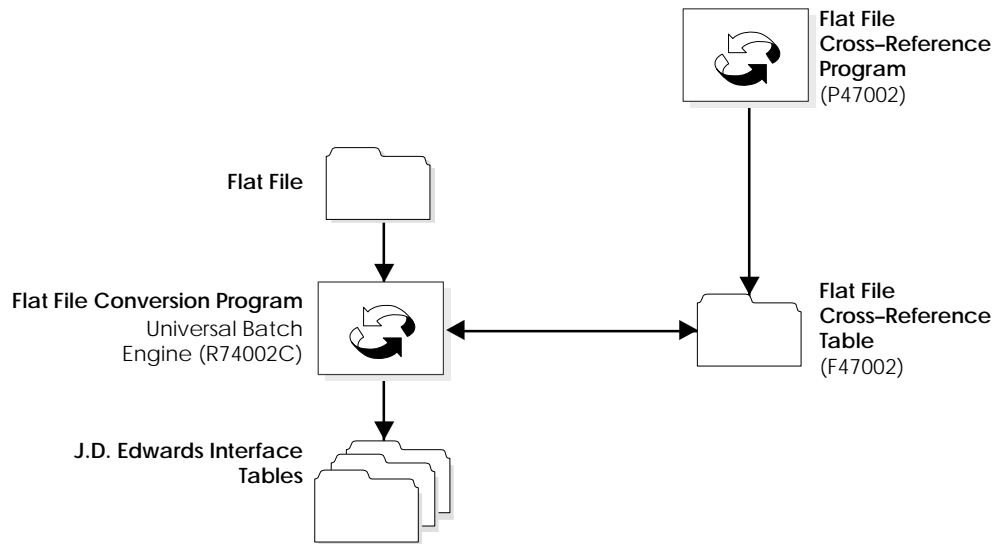


4. On Revisions, review and revise the transaction, and click OK.
5. If applicable, choose Detail Revisions from the Row menu to review or change additional detail information, and click OK when finished.

After you correct the errors identified by the Inbound Transaction Process, run the transaction process again. If other errors are identified, correct them and run the transaction process again.

Field	Explanation
User ID	The source of the transaction. This can be a user ID, a workstation, the address of an external system, a node on a network, and so on. This field helps identify both the transaction and its point of origin.
Batch Number	The number assigned to the batch. During batch processing, the system assigns a new batch number to the J.D. Edwards transactions for each control (user) batch number it finds.
Transaction Number	This is the number that an Electronic Data Interchange (EDI) transmitter assigns to a transaction. In a non-EDI environment, you can assign any number that is meaningful to you to identify a transaction within a batch. It can be the same as a J.D. Edwards document number.

Import from Flat Files



You use a flat file conversion program to import flat files into J.D. Edwards interface tables. The user can create a separate version of the Inbound Flat File Conversion program (R47002C) for each interface table. The Conversion program recognizes both the flat file it is reading from and the record types (record type user defined code table) within the flat file. Each flat file contains records of differing lengths based on the interface table record they correspond to. The Conversion program uses the Flat File Cross Reference table (F47002) to convert the flat file into the interface tables. The Flat File Cross Reference table indicates to the Conversion program which flat file to read from based on the transaction type you are receiving.

The Conversion program reads each record in the flat file and maps the record data into each field of the interface tables based on the text qualifiers and field delimiters specified in the flat file.

The Conversion program inserts the field data as one complete record in the interface table. If the Conversion program encounters an error while converting data, it withholds the data in error and continues conversion processing. If the data is successfully converted, the Conversion program automatically runs the Inbound Processor batch process for that interface table, if you set the processing options in the Conversion program to do so. For more information about error checking, see *Checking for Errors*.

You can use the OneWorld Inbound Processor to process the data once it is in the interface tables. If the Conversion program successfully converts all data into the interface tables, it automatically deletes the flat file after the conversion.

Setup Requirements for Flat File Conversion

If you use a flat file conversion, every field in the interface tables must be written to, even if the field is blank. For EDI, the translator software does this. The translator software used for EDI must be able to create a flat file, create fields, and put in delimiters. The default text qualifier is a double quote (“ ”) and the default field delimiter is a comma (.). However, any field delimiter and text qualifier may be used as long as they do not interfere with the interpretation of the fields. Use the processing options on the Conversion program to define what text qualifiers and field delimiters you will use.

If you are receiving documents with decimal numbers, you must use a place holder (such as a period) to indicate the position of the decimal. You define the place holder in the User Preference table. See the *OneWorld Foundation Guide* for more information about user preferences.

The first field value in a flat file record indicates the record type. In other words, the first field value indicates to the Conversion program which interface tables to insert the record. Record type values are defined and stored by the record type user defined code table 00/RD.

The format of the record in the flat file must follow the format of the interface table. This means that every column in the table must be in the flat file record and that the columns must appear in the same order as in the table.

For example, a record in the header table looked like the following:

Record Type	Name	Address	City	Zip Code
1	Joe	<Blank>	Denver	80237

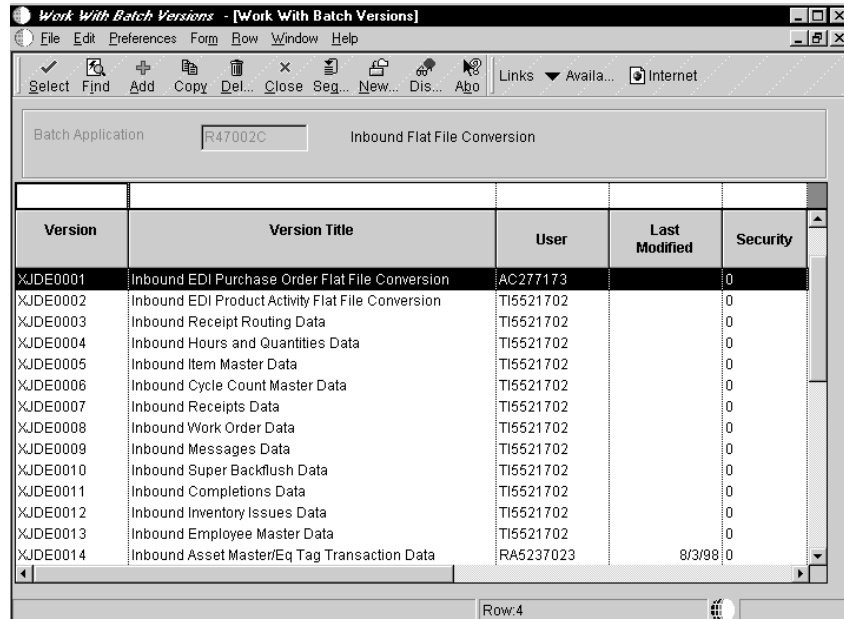
The record in the flat file would look like the following.

“1”, “Joe”, “ ”, “Denver”, ”80237”

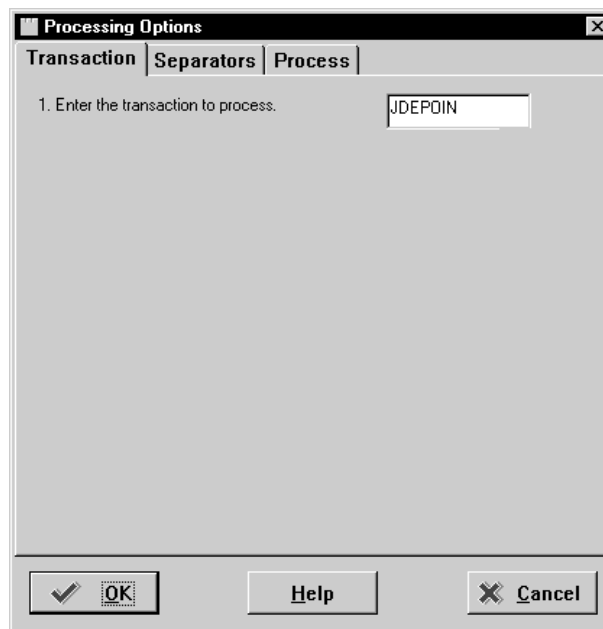
Notice that 1 that corresponds to a header record type and the blank space that corresponds the the <Blank> in the Address column.

► **To import from flat files**

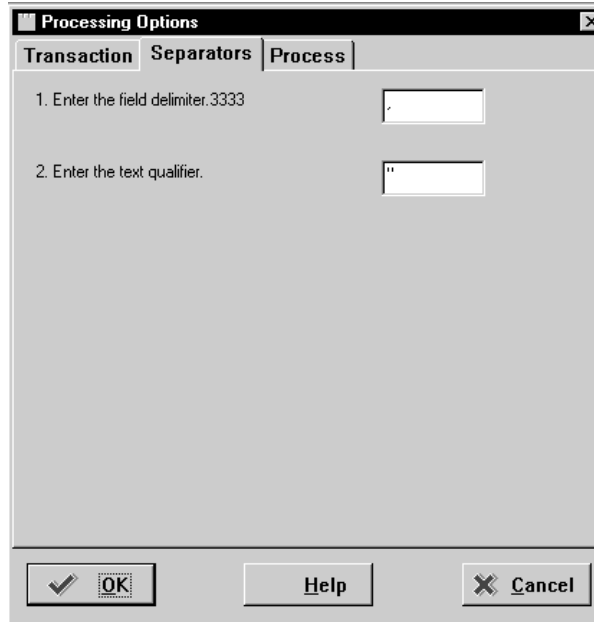
Open the Flat-File Conversion (R47002C) batch process.



1. On Work With Batch Versions, choose the program version that you want to use.
2. From the Row menu, choose Processing Options.

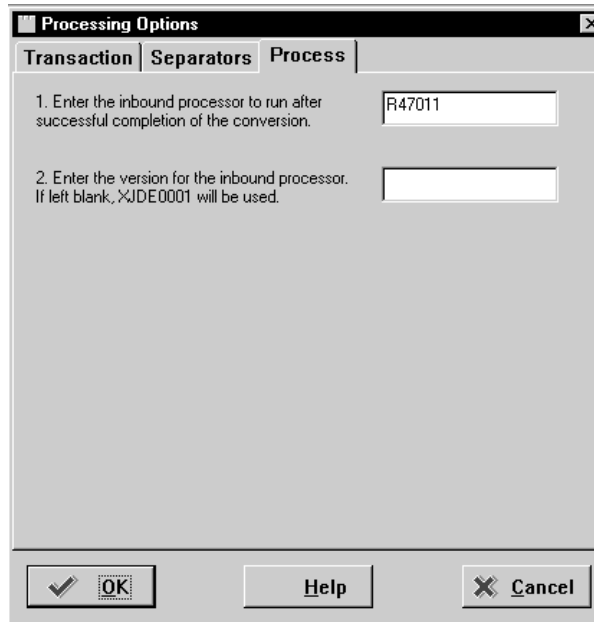


3. Click the Transaction tab and enter the transaction type that you are importing, for example, "JDEPOIN."



The screenshot shows the 'Processing Options' dialog box with the 'Transaction' tab selected. The dialog has three tabs: 'Transaction', 'Separators', and 'Process'. Under the 'Transaction' tab, there are two input fields. The first is labeled '1. Enter the field delimiter.3333' and contains a comma character. The second is labeled '2. Enter the text qualifier.' and contains a double quote character. At the bottom of the dialog are three buttons: 'OK' (with a checkmark icon), 'Help', and 'Cancel' (with an 'X' icon).

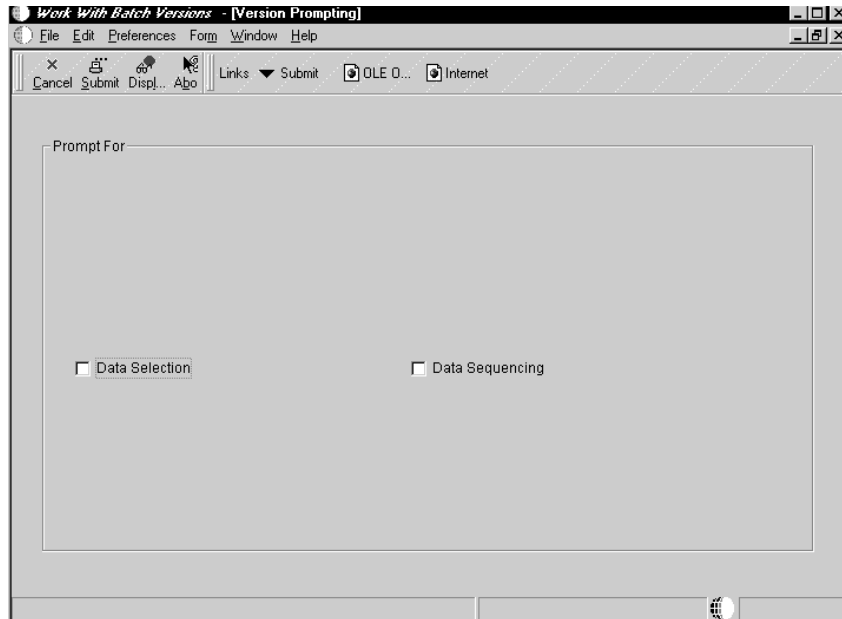
4. Click the Separators tab and enter the field delimiter character and the text qualifier character your system uses to identify fields and text.



The screenshot shows the 'Processing Options' dialog box with the 'Separators' tab selected. The dialog has three tabs: 'Transaction', 'Separators', and 'Process'. Under the 'Separators' tab, there are two input fields. The first is labeled '1. Enter the inbound processor to run after successful completion of the conversion.' and contains the text 'R47011'. The second is labeled '2. Enter the version for the inbound processor. If left blank, XJDE0001 will be used.' and is currently empty. At the bottom of the dialog are three buttons: 'OK' (with a checkmark icon), 'Help', and 'Cancel' (with an 'X' icon).

5. Click the Process tab and enter the name of the inbound program and the version of the program to run after the Flat File Conversion program has successfully completed.
6. Click OK.

7. On Work With Batch Versions, click Select.



8. On Version Prompting, click either of the following to review the report feature options:
 - Data Selection
 - Data Sequencing
9. Click the Submit button.



Detailed Tasks for Custom Programming

The interoperability tasks you need to perform are based on the method of access you choose to use. You may need to use custom programming to:

- Process Outbound Transactions
- Process Inbound Transaction Confirmation



Process Outbound Transactions

After outbound records are written to the interface tables, they must be processed by either a third-party function or a OneWorld batch process. The function or batch process can be written to your specifications, but it must accept the input parameters defined below. When the function or batch process has finished processing and is successful, it calls the Interoperability Outbound Confirmation Function API. This API marks the processing log record as successfully processed. This API can either be called from the third-party function, the outbound batch process, or it can be called from another object in the third-party's system process. See *Call OneWorld Completion Confirmation API* for more details.

Vendor-Specific Outbound Batch Processes

The purpose of the vendor-specific outbound batch process is to process unedited transaction table records in a batch mode, either in a batch-of-one or in true batch mode. The batch process is called by the generic outbound subsystem batch process, and receives the key to the record in the interface table.

Each vendor-specific batch process is specific to the transaction being processed. You must decide what the batch process actually does with the database record information. The batch processes are written to your specifications using the J.D. Edwards toolset. However, you must use the J.D. Edwards defined data structure shown below.

Data Item	Required	I/O	Description
EDUS	Y	I	User ID
EDBT	Y	I	Batch Number
EDTN	Y	I	Transaction Number
FFEM	N	I	Flat File Export Mode
EDEM	N	I	External Database Export Mode
EAEM	N	I	External API Export Mode
ERRC	N	O	Error Code
EDLN	N	I	Line Number

Vendor-Specific Outbound Functions

The purpose of the vendor-specific outbound function is to pass the key fields for a record in the OneWorld outbound unedited transaction tables to a third party. With these keys, you can process information from the database record

into the third-party system. The function is called by the generic Outbound Subsystem batch process.

Each vendor-specific function is specific to the transaction being processed. You must decide what the function actually does with the database record information. Although the functions are written to your specifications and most likely are written outside of OneWorld, these functions must use the required J.D. Edwards defined data structure shown below.

Data Item	Required	I/O	Description
szUserId	Y	I	User ID - 11 characters
szBatchNumber	Y	I	Batch Number - 16 characters
szTransactionNumber	Y	I	Transaction Number - 23 characters
mnLineNumber	Y	I	Line Number – double
szTransactionType	Y	I	Transaction Type - 9 characters
szDocumentType	Y	I	Document Type - 3 characters
mnSequenceNumber	Y	I	Sequence Number – double

Process Inbound Transaction Confirmation

You can provide a confirmation function to alert a third party that a transaction has been processed. The confirmation function also provides notification about whether the transaction processed successfully. The confirmation function is specific to a process and must accept the following parameters:

User ID	11 characters
Batch Number	16 characters
Transaction Number	23 characters
Line Number	double
Successfully Processed	1 character

The first four parameters are the key to the processed transaction. The last parameter is a success/fail flag. The library containing the function must be located on the same server that processes the inbound records.

To request inbound transaction confirmation, the name of the function and the full path of the library containing the function must be passed to the subsystem batch process that processes the transaction. This information is passed through the Inbound Transaction Subsystem data structure. See *Placing Entries in the Subsystem Data Queue* for more details on this data structure.

After the subsystem batch process finishes processing the transaction, it calls the inbound confirmation function, passing the key to the processed transaction and the notification about whether the transaction was successfully processed. The third party must decide how the confirmation function uses this information.



Additional Information

There is additional information that is useful for you to know in order to complete some tasks. The following topics provide additional information:

- Formatting Data
- Using Scheduler



Formatting Data

When data is sent from external systems into OneWorld, either through the Interface Table or by directly calling the master business functions, it must be formatted into OneWorld data types. Two of the most commonly used OneWorld data types are:

- MATH_NUMERIC
- JDEDATE

It is possible that these data types may change. For that reason, it is critical that you use the Common Library APIs provided by OneWorld to manipulate variables of these data types.

MATH_NUMERIC Data Type

There are no numeric values stored within the OneWorld database. Numeric values are stored as a MATH_NUMERIC data type. A MATH_NUMERIC is a structure that contains all the required information to execute mathematical operations on a stored value. This data type allows the attributes of a number to be stored directly with the value, providing greater flexibility when formatting the number for display.

```
Struct tagMATH_NUMERIC
{
    char    String [MAXLEN_MATH_NUMERIC + 1];
    char    Sign;
    char    EditCode;
    short   nlength;
    short   nLength;
    WORD    wFlags;
    char    szCurrency [4];
    short   nCurrencyDecimals;
    short   nPrecision;
};

typedef struct tagMATH_NUMERIC NATH_NUMERIC, FAR *LPMATH_NUMERIC;
```

MATH_NUMERIC Element	Description
String	The digits without separators.
Sign	A minus sign indicates the number is negative, otherwise the value is 0x00.
EditCode	The data dictionary edit code used to format the number for display.
nDecimalPosition	The number of digits from the right to place the decimal.
nLength	The number of digits in the String.
wFlags	Processing flags.
szCurrency	The currency code.
nCurrencyDecimals	The number of currency decimals.
nPrecision	The data dictionary size.

MATH_NUMERIC APIs from the Common Library

Following are commonly used OneWorld MATH_NUMERIC APIs:

DoubleToMathNumeric	Copies the numeric value of a variable of type double to a variable of type math numeric.
FormatMathNumeric	Formats a JDE MATH_NUMERIC into a fully formatted string, ready for output.
IncrementMathNumeric	Increments or decrements a referenced MATH_NUMERIC by one.
IntToMathNumeric	Converts an integer to a MATH_NUMERIC.
LongToMathNumeric	Converts a long to a MATH_NUMERIC.
MathAdd	Adds parameter two to parameter three and returns the sum in parameter one. All parameters are MATH_NUMERIC data types.
MathCompare	Returns a "1" if parameter one is greater than parameter two; returns a -1 if parameter two is greater than parameter one. Both parameters are MATH_NUMERIC data types.
MathCompareAbsolute	Same as MathCompare, except that it will compare the absolute value of the parameters, for example, -123 would be greater than +100.
MathCopy	Copies the value of a source MATH_NUMERIC value (parameter two) to a destination MATH_NUMERIC value (parameter one).
MathDivide	Divides parameter three by parameter four and places the quotient in parameter one. The remainder is placed in parameter two. All parameters are MATH_NUMERIC data types.
MathMultiply	Multiplies parameter two by parameter three and places the result into parameter one. All parameters are MATH_NUMERIC data types.
MathNumericToDouble	The function copies the numeric value of a MATH_NUMERIC variable to a variable of type double.
MathNumericToInt	Converts a MATH_NUMERIC into an integer (int). For example, when you pass in 1.9, it is treated as one.
MathNumericToLong	Converts a MATH_NUMERIC to a long (Checks that MATH_NUMERIC does not have decimals).
MathRound	Rounds a MATH_NUMERIC passed in parameter two to the specified number of digits passed in parameter three. The result is returned in parameter one.
MathStringToLong	Translates a number string into a long number. The value is passed as a return code.
MathSubtract	Subtracts parameter three from parameter two and places the result into parameter one. All parameters are MATH_NUMERIC data types.

MathZeroTest	Returns an int 0 if the string portion of the struct is equal to zero, and the length is one.
ParseNumericString	Formats a numeric string into the MATH_NUMERIC format.
ReverseMathNumeric	Reverses the sign on a MATH_NUMERIC.
ZeroMathNumeric	Sets a MATH_NUMERIC to zero.

JDEDATE Data Type

All dates in the OneWorld database are stored in a JDEDATE structure to allow the greatest flexibility for formatting the dates.

```
Struct tagJDEDATE
{
    short    nYear;
    short    nMonth;
    short    nDay;
};

typedef struct tagJDEDATE JDEDATE, FAR *LPJDEDATE;
```

JDEDATE Element	Description
nYear	The year (four digits).
nMonth	The month.
nDay	The day.

JDEDATE APIs from the Common Library

Following are some commonly used OneWorld APIs:

AdvanceDate	Takes in a date in JDEDATE form, a number of months, a number of days and returns the advanced (decremented) date in JDEDATE form.
AdvanceOneDay	Increments the day, in Gregorian form, by one day.
AdvanceOneMonth	Increments the month, in Gregorian form, by one month.
CalcN	Takes in a Gregorian date and calculates a long integer to represent that date as an offset from 01/01/0000.
DateDifference	Returns a long integer value that is the difference between two dates.
DayOfTheWeek	Returns the day of the week based upon the N value of a date.
DecrementOneDay	Decrements the day, in Gregorian form, by one day.
DecrementOneMonth	Decrements the month, in Gregorian form, by one month.
DeformatDate	Changes a date string into the JDEDATE format.
DerefN	Takes in a long integer that represents a date as an offset from 01/01/0000 and calculate its corresponding Gregorian date.
FormatDate	Formats an internal JDEDATE into a formatted string.
IsJDEDATENull	Tests whether a JDEDATE is NULL.
IsLeapYear	Determines if the year is a leap year, based upon a Gregorian date.
JDEDATEToCentury	Gets the century from a JDEDATE and returns it in a Math_Numeric.

JDEDATEToCenturyAndYear	Gets the year and century from a JDEDATE and returns them in Math_Numeric.
JDEDATEToday	Formats a JDEDATE with today's date.
JDEDATEToYear	Gets the year from a JDEDATE and returns it in a Math_Numeric.
NumberOfDays	Finds the number of days in a month based upon a Gregorian date.

Strings and Characters

Strings and characters in OneWorld do not require special data types or APIs for manipulation. They may be initialized and set to a value using the standard functions provided by programming languages. OneWorld string sizes allow for a null terminator. For example, if the string size is 11, there is enough space for ten characters and a null terminator.

Unused Structure Members

When you directly call master business functions or add records to interface tables, unused members of the data structures may be left as NULL. The JDEBASE APIs will translate the null values into zero for MATH_NUMERIC and JDEDATE fields and blanks for string and character fields.

Special Circumstances

Certain OneWorld fields require special formatting prior to adding them to the interface tables or master business function data structures. Two commonly used fields are MCU and KCOO. MCU is a right-justified string, padded with leading blanks. KCOO is a right-justified string padded with leading zeros. Use the OneWorld Data Dictionary to check for other fields that may require special formatting.

Using Scheduler

You might want to run subsystem jobs that take up a great deal of machine resources or that require users to be signed off of the software after normal working hours. You might also want the flexibility of running jobs at scheduled intervals during the day.

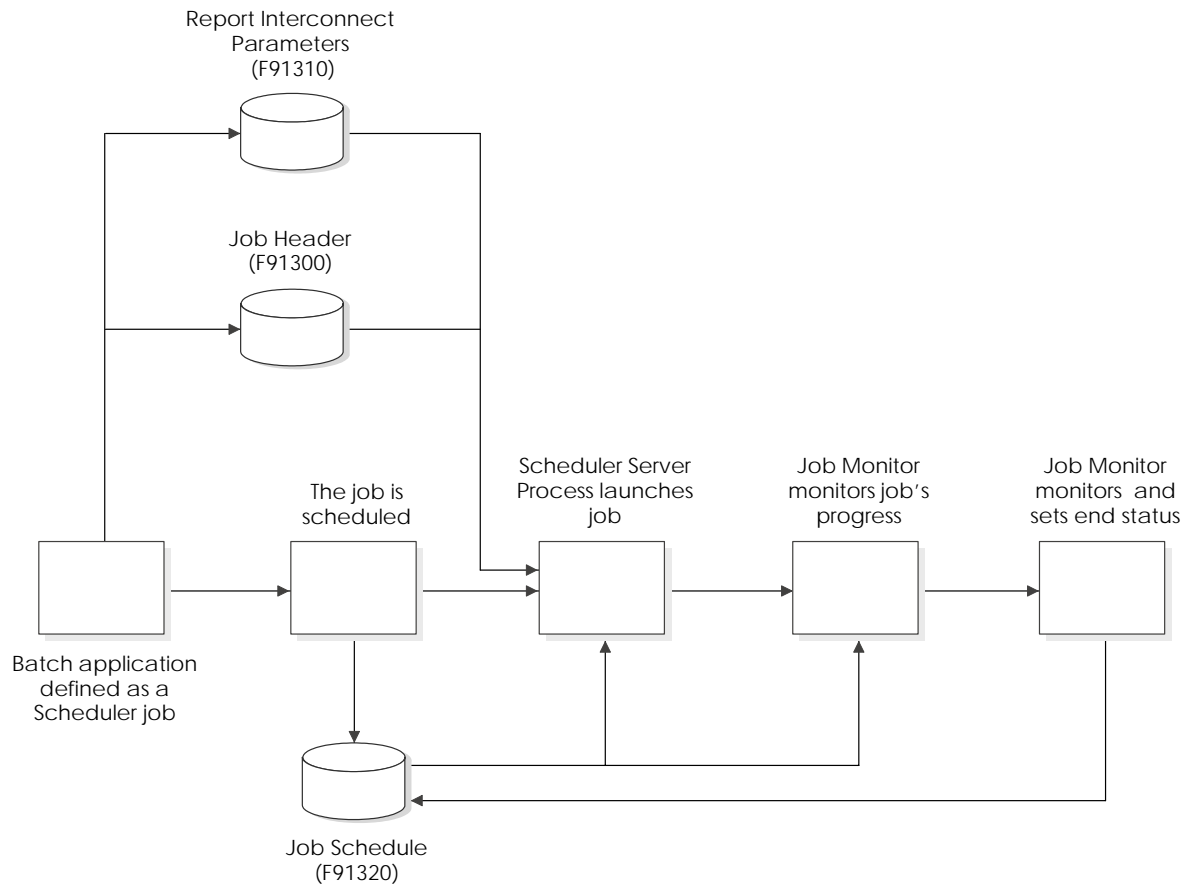
OneWorld Scheduler allows you to schedule subsystem jobs to run after hours or throughout the day, if necessary. The ability to run jobs at specific times allows you the flexibility of running jobs without impacting the next day's schedule. You can schedule jobs based on day, day of year, time, job completion, day of period, and day of week. You can also set up Scheduler to restart a job in the event of a job failure.

You can even specify on which server you want the job to run and in which time zone, whether it is in your city or in some other country. Scheduler uses a modified version of Universal Time Coordinate (UTC) time, which counts the number of minutes, not seconds.

After the job is scheduled, Scheduler writes records to the Job Schedule table (F91320), indicating each time that the job will be launched. As the job runs, the Job Monitor monitors the job's progress.

When the job ends, the Job Monitor sets an end status to the job and updates the job's record in the Job Schedule table (F91320) to indicate that the job either ended successfully or in error.

The following graphic illustrates the scheduling process. When you define a scheduled job, the parameters of that job are stored in the Job Master table (F91300).



See Also

- *Scheduler* in the *System Administration Guide* for more information about scheduling jobs

Appendices



Appendix A - Interoperability Features Created by J.D. Edwards

J.D. Edwards has used some of the technologies and models described in this guide to create interoperability features. The information in this appendix describes one method you can use. This method uses flat files, database files, and master business functions.

These predefined features make it easier for third parties to interface with OneWorld. You can use similar procedures to create your own interoperability features. You can use the following interoperability features and standards to shorten design time, provide consistency, and provide easier maintenance. This section provides information and standards for the following features:

- Transactions Into and From OneWorld
- Transactions Into OneWorld
- Transactions From OneWorld

Transactions Into OneWorld

There are three different supported entry points into the J.D. Edwards system using this method:

- flat file (ASCII text file)
- database table
- Master Business Function (MBF) interactive call

The format of all data being imported from external systems must be presented in the J.D. Edwards defined format for the chosen mode of interface.

Inbound Through a Flat File

An external source creates a flat file in the J.D. Edwards specified format for the transaction or master file. The flat file format uses standard delimiters for different data elements.



Unedited inbound data in the flat file is converted into records in J.D. Edwards database tables (unedited transaction tables). Conversion of the flat file data into J.D. Edwards database table records may be performed using existing flat file to database conversion utilities or by running an inbound conversion batch process. A generic business function within the batch process maps the flat file into the database tables.

The unedited transactions tables are processed by an Inbound Processor batch process to call the appropriate Master Business Function to update the J.D. Edwards live data.

If required, a preprocessor business function can be run from the Inbound Processor batch process to establish key information matching the unedited transaction record to the original application record, for example, a key to a cash receipt or purchase receipt.

If the master business function returns errors for the transaction, the appropriate record is flagged on the audit report and errors are sent to the message center in the form of action messages. These action messages, when invoked, call a revision application that allows modification of the unedited transaction data directly.

Transactions that have been successfully updated to the live files are flagged as successfully processed in the unedited transaction tables.

Inbound Transactions to Unedited Transaction Tables

Inbound transactions may be written directly to the unedited transaction tables that are already in a database format. The unedited transaction tables are processed by an Inbound Processor batch process to call the appropriate Master Business Function to update the J.D. Edwards live data.

Optionally, the Inbound Processor batch process may be run in a subsystem. The key information written to the unedited transaction table is added to the subsystem data queue. This triggers the Inbound Processor batch process that processes that specific key.

A user supplied confirmation function can be executed after the processor batch process completes. The purpose of the confirmation function is to alert the third party that a transaction that they sent into the J.D. Edwards system has been processed. They are also notified whether the transaction was processed successfully. These confirmation functions are specific to a process and thus are only given the keys to the transactions (EDUS, EDBT, EDTN, EDLN) and the successfully processed flag. The third party must decide what the confirmation function actually does with this information. These vendor-specific interoperability inbound confirmation functions are called from the Inbound Processor batch program's through the Call Vendor-Specific Function - Inbound business function. The confirmation functions are written by third parties to their own specifications are most likely written outside of OneWorld. However, they use a J.D. Edwards defined data structure.

If required, a preprocessor will run from the Inbound Processor batch process to establish key information matching the unedited transaction record to the original application record and so on, for example, the key to a cash receipt or purchase receipt.

If the master business function returns errors for the transaction, the appropriate record is flagged on the audit report and errors are sent to the message center in the form of action messages. These action messages, when invoked, call a revision application that allows modification of the unedited transaction data directly.

Transactions that have been successfully updated to the live files are flagged as successfully processed in the unedited transaction tables.

Inbound Transactions Passed Directly to Master Business Functions

Inbound transactions may be passed directly to the J.D. Edwards master business functions for processing. No data is stored in the unedited transactions tables.

The master business function is called directly through support from JDENET. Certain J.D. Edwards internal APIs must be used to call the master business function directly. The master business function updates the live data if no errors are found.

Messages are added to the messages linked list in the event of errors. It is the responsibility of the calling program to handle these exceptions

Transactions From OneWorld

Transactions that need to be exported to external systems are processed in a manner similar to the inbound transactions, except in reverse. The logic to determine whether a transaction needs to be interfaced to another system is set up in a separate table, the Data Export Control table. In that event, the master business function will handle logging all adds, changes, and deletes to the J.D. Edwards live data to the unedited transaction tables.

Data is formatted to the specifications of the connecting system, be that through the use of a flat file, a third-party database, or an external API call. This export will be done by vendor-specific batch processes.

Common Pieces of the Outbound Process

An application or batch process enters data and calls a master business function to update the J.D. Edwards database.

Master business function (EditLine) determines whether an interface to an external system is applicable by checking the value of a processing option that

contains a transaction type. If the processing option is not blank, the master business function will be interfacing to an external system.

During the EndDoc processing when the transaction is being written to the J.D. Edwards database, records are also written to the unedited transaction tables. For inquiries, additions, and deletions of a J.D. Edwards database record, one record is written to an unedited transaction table. For changes to a J.D. Edwards database record, two records can be written to an unedited transaction table. These records contain an image of the database record before the change and an image of the database record after the change. A processing option determines whether the before image record is written.

After a transaction has been written, a key is sent from the master business function process to the subsystem data queue to trigger processing of a newly added record in the unedited transaction table. The subsystem data queue will launch an outbound subsystem batch process. A generic outbound subsystem batch process handles all outbound transactions.

The outbound subsystem batch process reads records from the Data Export Control table to determine the vendor-specific business function or batch process to run. The vendor-specific business function must be written to J.D. Edwards specifications relative to the data structure and the vendor defines its name and location within the Data Export Table. A vendor-specific batch process can also be defined in the Data Export table. The outbound subsystem batch process recognizes whether a function or batch process has been requested. A processing log record is written for tracking purposes. If the function method is requested, the specified function is called real-time within the subsystem batch process. If the batch process method is requested, it calls the batch process only if the delayed option is not specified. To complete the processing for the delayed option, the user must set up the appropriate batch process in OneWorld Scheduler to process the correct records in the processing log table as often as required.

Knowledge of pending outbound transactions is retained in persistent storage until the third party overtly indicates to J.D. Edwards that they have successfully processed the transaction. J.D. Edwards provides an outbound confirmation function that the third party must use to confirm success. This applies to both the batch process and function methods. The third party can process real-time or defer to any downstream process. When the third party completes their process, they need to call this outbound confirmation function to let J.D. Edwards know that we can update the transaction as successful and make it eligible for purging.

The outbound subsystem batch process tracks the transactions processed, the vendor-specific functions and batch processes that are run, and whether the functions and batch processes were successful. This information is stored in the Processing Log table. If all vendor-specific batch processes were successful, the records in the unedited transaction tables for the transaction are updated as successfully processed.

The Processing Log table contains a foreign key back to the data Export Control table. This enhances the recovery capabilities in that a program can look at a process control record and by looking back at the associated data export control record, can know what would be required to reprocess it.

WorldSoftware Coexistence

When a client is running OneWorld in coexistence mode, transactions may be entered either through a OneWorld application or a WorldSoftware program. For data export to external systems to work in this configuration, WorldSoftware programs need to write to the unedited transaction tables.

Database triggers, written for the AS/400 database in RPG-ILE, test to determine whether data needs to be exported. When it does, they write to the unedited transaction tables.

Net Change Considerations

Changing a set of data elements of a transaction to new values (including the value of blank) is referred to as “Net Change Processing”.

Net change is handled by the deliberate use of NULL. A NULL value in a field indicates to the master business function that this inbound field is not to be changed from its current value. Any other value in the inbound field is validated and subsequently updated to the database.

To handle net change in this manner there are some considerations:

- When a record is inserted to a database, any fields that are NULL are initialized to one blank. For the unedited transaction tables, an additional characteristic is required whereby the database middleware does not initialize a NULL value to blank. This allows a table column to have a value of NULL, which can be assigned to a Master Business Function data structure parameter.
- Currently the OneWorld Development Tools initialize data structures to blanks when a form is initialized. A system function is needed whereby a data structure can be initialized to NULL. This would allow net change logic to be used in event rules and named event rules by permitting them to initialize a data structure to NULL.
- You can initialize controls and variables to assign a value of NULL. This allows event rules and named event rules to assign NULL to controls and variables before passing the values to a master business function data structure.

Information Structure

Information used in the Interoperability module can be divided into master file maintenance and transaction processing. The file layouts of these files can be found in the online documentation. The following information is available.

- Master files
- Data export control

This table contains records that specify the vendor-specific functions, batch processes, and batch versions to run for a transaction and document type combination. It also contains information on which types of records (adds, updates, etc.) are added to the unedited transaction tables.

- Flat File Cross-Reference Table

This table contains cross-reference information about a transaction. This information will be flat file name, direction of transaction, application tables used by the transaction, and the record types of the application tables.

- Transaction files
- Processing log

This table contains information on whether vendor-specific functions and batch processes processed a transaction successfully.

Creating Transactions Into and From OneWorld

This chapter defines the standards that are common to creating both inbound and outbound transactions, including standards for the following:

- Transaction name
- Unedited transaction tables
- Revision application
- Purge named event rules and batch processes
- Subsystem business function

Transaction Name

The transaction name must be defined in user defined code 00/TT in the Pristine environment. The name starts with “JDE” and can be up to eight characters in length. The following examples illustrate a proper transaction name:

- JDERR for Receipt Routing Transaction
- JDEWO for Work Order Header Transaction

Unedited Transaction Tables

Each transaction has its own set of tables, except when an existing set of tables meets the needs of the transaction. “Meeting the needs of the transaction” means that the existing set of tables contains all of the required fields for the new transaction. These existing tables must follow interoperability standards. They should be Z1 tables for interoperability, not EDI tables.

One set of tables is used by both the inbound and the outbound directions of an internal transaction within a system. For example, in the Sales Order system, for a sales order, the inbound Customer Order (850) and the outbound Order Acknowledgment (855) share a set of tables under this standard.

Use the following guidelines to determine the based-on table:

- Inbound is based on the application table that is updated with data from the unedited transaction table

- Outbound is based on the application table that has data extracted from it into the unedited transaction table

If the unedited transaction table is used for both inbound and outbound transactions, the based-on table should be the same application table. In the above Sales Order example with an inbound Customer Order and an outbound Order Acknowledgment, the detail unedited transaction table would be based on the F4211 table.

The unedited transaction table is named after the based-on table with “Z1” as the suffix. For example, if the application table is F4211, the unedited transaction table would be F4211Z1. Subsequent transaction tables based on the same application table will have Z2, Z3, and so on as the suffix.

If the unedited transaction table exceeds 250 columns or has a record length greater than 1968, an additional transaction table is needed for the remaining columns. Columns in the additional transaction table should contain infrequently used data. The additional transaction table is named after the primary transaction table with a letter, starting with A, after the Z1 suffix. For example, if the primary unedited transaction table is F4211Z1, the additional table would be F4211Z1A.

The beginning of the table has the following columns, which act as control fields:

- User ID (EDUS) - key field
- Batch Number (EDBT) - key field
- Transaction Number (EDTN) - key field
- Line Number (EDLN) - key field
- Document Type (EDCT)
- Transaction Type (TYTN)
- Translation Format (EDFT)
- Transmission Date (EDDT)
- Direction Indicator (DRIN)
- Number of Detail Lines (EDDL)
- Processed (EDSP)
- Trading Partner ID (PNID)
- Action Code (TNAC)

You must use the key structure above.

The following tables show an example of the use of Line Number for net change processing, where UB stands for the image of the record before the update and UA stands for the image of the record after the update. This example ignores table layout standards.

Header Table

User ID	Batch Number	Transaction Number	Line Number	Action	Order Number
TI	77	100	1	UB	2000
TI	77	100	2	UA	2000

Detail Table

User ID	Batch Number	Transaction Number	Line Number	Action	Order Number	Order Line
TI	77	100	1	UB	2000	1
TI	77	100	2	UA	2000	1
TI	77	100	3	UB	2000	2
TI	77	100	4	UA	2000	2

The end of the table has columns that are reserved for user and audit fields.

- User Reserved Code (URCD)
- User Reserved Date (URDT)
- User Reserved Amount (URAT)
- User Reserved Number (URAB)
- User Reserved Reference (URRF)
- Transaction Originator (TORG)
- User ID (USER)
- Program ID (PID)
- Work Station ID (JOBN)
- Date Updated (UPMJ)
- Time of Day (TDAY)

The middle of the table has all of the columns from the based-on application table, excluding user reserved and audit field columns. An exception to this is when the transaction table is near the 250-column limit or the 1968-record length limit. In this case, columns from the application table that most likely will not be needed should be excluded.

Prefixes for the table columns are SY for the header and SZ for the detail.

Change/match transaction tables, such as a cash receipt or purchase receipt, might require additional columns that correspond to user input capable controls on an interactive form.

A header table is not required for every transaction.

Revision Application

This application is used to correct data and to add new transaction records. The application will call the purge NER to do deletions of records. The name is based on the detail unedited transaction table.

For example, if the tables for Sales Order Entry were F4201Z1 and F4211Z1, the application would be called P4211Z1.

Purge Batch Process and Named Event Rules

The purge batch process should have one or two sections. The number of sections depends on the unedited transaction tables. The purge batch process calls the purge NER. The name of the purge batch process is based on the revisions application with a P suffix. For example, if the revisions application is P4211Z1, the purge batch process is R4211Z1P.

Purge named event rules has two modes:

- Header mode, which deletes the header record and all associated records independent tables.
- Detail mode, which deletes the detail record and all associated records in dependent tables.

The purge NER is named after the purge batch process. Only eight characters are allowed for the NER name. If the name has nine characters using these standards, remove the P suffix. For example, if the purge batch process is R4211Z1P, the purge NER will be N4211Z1P.

When a “before image” for net change is deleted, the corresponding “after image” is also deleted. When an “after image” is deleted, the corresponding “before image” is also deleted.

Subsystem Business Function

There are two generic subsystem business functions *Add Inbound Transaction To Subsystem Queue* (B0000176) for inbound and *Add Transaction To Subsystem Queue* (B0000176) for outbound transactions. These functions write a record to the subsystem data queue to specify a batch process that needs to be awakened in the subsystem.

The business function starts processing of a Batch of One (single transaction). The business function also passes keys to the subsystem data queue.

The data structure for the inbound transaction is the batch process name (OBNM), version (VERS), user ID (EDUS), batch number (EDBT), transaction number (EDTN), line number (EDLN), suppress error message (EV01), confirmation function name (FCNN), and confirmation function library (FCNL).

The data structure for the outbound transaction will be line number (EDLN), transaction type (TYTN), document type (DCTO, and action code (TNAC).

Creating Transactions Into OneWorld

This chapter contains information specific to creating inbound transactions. It refers to common standards for creating inbound and outbound transactions. It includes information about the following topics:

- Inbound processor batch process
- Inbound conversion batch process
- Inbound flat file

Refer to *Transactions Into and From OneWorld* for more information about these standards.

Creating an Inbound Transaction

To create an inbound transaction complete the following steps:

1. Name your transaction.

Refer to *Creating Transactions Into and From OneWorld* for more information about these standards.

2. Create the inbound unedited transaction tables.

Refer to *Creating Transactions Into and From OneWorld* for more information about these standards.

3. Create a revision application.

Refer to *Creating Transactions Into and From OneWorld* for more information about these standards.

4. Create an inbound processor batch program.

This batch process calls a master business function to edit data and update application tables. It might need modifications for Source of Data and Change Logic.

Errors are sent to the Message Center. An audit report listing transactions in error and a single line stating the total number of transactions that were successfully processed.

The batch process can have a processing option for an automatic purge. The batch process may need preprocessors to match transaction records to application records.

The data structure of the batch process, which gets populated by the subsystem, will be user ID (EDUS), batch number (EDBT), transaction number (EDTN), line number (EDLN), function name (FCNN), and function library (FCNL).

If a function name and a function library were passed into the data structure through the subsystem, call the Inbound Vendor-Specific Confirmation Function (B0000192).

The batch process should have a version that is called from the menu and a version that is called through the subsystem. The name of the batch process is based on the detail unedited transaction table with the letter I (for Inbound) as a suffix. For example, if the tables for Sales Order Entry were F4201Z1 and F4211Z1 the batch process would be called R4211Z1I.

5. Create a purge batch process and named event rule over the inbound unedited transaction tables. Refer to Transactions Into and From OneWorld.
6. Create a new version of the inbound conversion batch process (R47002C) for your transaction.

There is one batch process, R47002C. Each transaction needs its own version of the batch process. New transactions must be added to UDC 00/TT.

7. Add transaction to Flat File Cross Reference Table.

Add your transaction and its unedited transaction tables to the Flat File Cross Reference Table *(F47002) in the pristine environment. You can use P47002 to do this. When you create the flat file, leave it blank so that it can be overwritten with data.

Inbound Flat File

The first field in a flat file record is the record type, which has a value determined by the record type UDC table 00/RD. The hard-coded values are:

- 1 - Header
- 2 - Detail
- 3 - Additional Header
- 4 - Additional Detail

5 - SDQ

6 - Address

7 - Header Text

8 - Detail Text

The format of the record in the flat file must follow the format of the table. This means that every column in the table must be in the flat file record and the columns must appear in the same order as in the table.

For example, suppose a record in the header table looked like the following (This example ignores table layout standards.):

Name	Address	City	Zip
Joe	<Blank>	Denver	80237

The “record” in the flat file would look like the following. Notice the “1” that corresponds to a header record type and the blank space that corresponds to the <Blank> in the Address column.

”1”,”Joe”,””,”Denver”,” 80237”

Any field delimiter and separator may be used as long as they do not interfere with the interpretation of the fields.

Dates must be in the format MM/DD/YY. Numeric fields must have a decimal as the place keeper. A comma should not be used.

If you are using a PC, you must be able to map a drive on the PC to the location of the flat file.

Creating Transactions from OneWorld

This chapter contains information about creating outbound transactions. It refers to common standards for creating transactions into and from OneWorld. It includes information about the following topics:

- Master business function (EndDoc)
- Outbound purge batch process
- Flat file cross reference

To create a transaction from OneWorld, complete the following tasks:

1. Name your transaction.

Refer to *Creating Transactions Into and From OneWorld* for more information about these standards.

2. Create the outbound unedited transaction tables.

Refer to *Creating Transactions Into and From OneWorld* for more information about these standards.

3. Create a revision application.

Refer to *Creating Transactions Into and From OneWorld* for more information about these standards.

4. Modify your Master Business Function EndDoc module.

EndDoc updates application tables.

EndDoc writes records to the unedited transaction tables based on the Transaction Type processing option.

The unedited transaction table has the key structure User ID (EDUS), Batch Number (EDBT), Transaction Number (EDTN), and Line Number (EDLN), which is populated by the master business function.

EDUS is the user ID for the user calling the MBF.

EDBT is the batch number populated by *EDI Batch Number, Get Next Number* (N4700060).

EDTN is the transaction number normally populated by *Get Next Transaction Number* (B0000175) unless the transaction needs another type of unique identifier in this key.

EDLN is the line number sequentially assigned by the MBF.

You must add two processing options to the template that your master business function uses. The tab for the Transaction Type processing option is called "Interop".

- The first processing option controls the launching of the generic outbound subsystem batch process and to determine the transaction type written to the unedited transaction table. The processing option reads: *1. Enter the transaction type for the interoperability transaction. If left blank, the outbound interoperability processing will not be performed.*
- The second processing option is to control the writing of the before image record. The processing option reads: *2. Enter a '1' to write the before image for a change transaction. If left blank, only the after image will be written.*

Before and after images of application records are written to the unedited transaction tables by EndDoc. Writing the before image is a performance hit, use processing option 2 to control whether the before image will be written. The after image will always be written for a change to the application table record.

EndDoc calls the subsystem business function *Add Transaction to Subsystem Queue* (B0000176) to write keys to the subsystem data queue. This triggers the processing of the transaction that was added to the unedited transaction tables. This will only happen if the interoperability processing option 1 is set to 1 to immediately launch the outbound batch process and write the keys of the outbound table.

5. Create a purge batch process and named event rule.

Each outbound purge batch process will purge a specific set of outbound interoperability tables. This batch process is run to keep the database tables to a manageable size. The batch process will be run from a menu selection and will have data processing to only select the records that have been successfully processed. The batch process will have two modes determined by the number of tables for the transaction (one table or two tables):

- Detail-only purge

In a detail-only purge over one transaction table, the detail record is selected only if it is marked as processed, which is data selection for Successfully Processed equal to "Y". If it is, the processing log table (F0046) is checked through the business function F0046, Check

Interoperability Processing Log Record Status (N0000181) to see if all F0046 records corresponding to the detail record have been marked as processed or have been bypassed. If this is true, the batch process deletes the detail record through its corresponding purge NER then calls F0046 Purge Interoperability Processing Log (N0000181) to delete the processing log records.

- Hierarchical purge

The hierarchical purge selects header records marked as processed. Next the processing log is checked through the business function, F0046 Check Interoperability Processing Log Record Status (N0000181), to make sure the corresponding records are all marked as processed or bypassed. If this is true, the F0046 purge interoperability processing log (N0000181) business function is called to delete the processing log records corresponding to the header record. Then another section retrieves the detail records and the F0046 Purge Interoperability Processing Log (N0000181) business function is called to delete the processing log records corresponding to each detail record. Then the corresponding purge NER is called to delete the header record and corresponding detail records.

6. Add transaction to flat file cross reference table

Add your transaction and its unedited transaction table to the flat file cross reference table (F47002) in the pristine environment. You can use P47002 to do this.

Task Summary for J.D. Edwards Interoperability Features

If you use the methods described in this section, the following table illustrates the steps you use for each of the interoperability models.

Task	IS	IA	IB	FS	FA	FB
Create a master business function.	X	X	X		X	
Determine a name for your transaction.		X	X		X	X
Create interface tables.		X	X		X	X
Create a revision application over the interface table.		X	X			
Create an inbound processor UBE over the interface table.		X	X			
Create a new version of the inbound conversion UBE (R47002C) for your transaction. (This is used for flat file conversions.)		X	X			
Add your transaction and interface tables to the Flat File Cross Reference Table (F47002) in the pristine environment.		X	X		X	
Include logic in your MBF to log changes.					X	
Create a retrieval API or business function.				X		
Create a Purge UBE and NER over the interface table.		X	X		X	X
Document the interface table format.		X	X		X	X

IS Into OneWorld Synchronous

IA Into OneWorld Asynchronous

IB Into OneWorld Batch

FS From OneWorld Synchronous

FA From OneWorld Asynchronous

FB From OneWorld Batch

Appendix B – Interoperability Interface Table Information

	Z-Table(s)	Input Subsystem Batch Process	Input Processor Batch Process	Extraction Batch Process	Revisions Application	Purge Batch Process	Application with Processing Options
Financials							
Address Book	F0101Z2	R01010Z – ZJDE0002	R01010Z – ZJDE0001		P0101Z1	R0101Z1P	P0100041
Customer Master	F0301Z21	R03010Z – ZJDE0002	R03010Z – ZJDE0001		P0301Z1	R0101Z1P	P0100042
Supplier Master	F0401Z1	R04010Z – ZJDE0002	R04010Z – ZJDE0001		P0401Z1	R0101Z1P	P0100043
A/R Invoice	F03B1Z1, F0911Z1, F0911Z1T	R03B11Z1I	R03B11Z1I – ZJDE0001	N/A	P03B11Z1	R03B11Z1P	N/A
A/P Invoice	F0411Z1, F0911Z1	R04110Z – ZJDE0002	R04110Z – ZJDE0001	N/A	P0411Z1	R0411Z1P	N/A
Payment Order with Remittance	F0413Z1, F0414Z1	N/A	N/A		P0413Z1	R0413Z1	P0413M
Journal Entry	F0911Z1, F0911Z1T	R09110Z – ZJDE0005	R09110Z – ZJDE0002		P0911Z1	R0911Z1P	N/A
Fixed Asset Master	F1201Z1, F1217Z1	R1201Z1I – XJDE0002	R1201Z1I – XJDE0001	R1201Z1X	P1201Z1	R1201Z1P	P1201
Account Balance	F0902Z1	N/A	N/A	N/A	P0902Z1	R0902ZP	N/A
Batch Cash Receipts	F03B13Z1	N/A	R03B13Z1I – ZJDE0001	N/A	N/A	N/A	N/A
HRM							
Payroll Time Entry	F06116Z1	R05116Z1I	R05116Z1I – ZJDE0001	N/A	P05116Z1	R05116Z1P	N/A
Distribution							
Purchase Order	F4301Z1, F4311Z1	R4311Z1I – XJDE0002	R4311Z1I – XJDE0001		P4311Z1	R4301Z1P	P4310
Outbound Purchase Receipts	F43121Z1	N/A	N/A		P43121Z1	R43121Z1P	P4312
Receipt Routing	F4309Z1	R4309Z1I – XJDE0002	R4309Z1I – ZJDE0001		P4309Z1	R4309Z1P	P43250
Outbound Sales Order	F4201Z1, F4211Z1, F49211Z1	N/A	N/A		P4211Z1	R4211Z1P	P4210
Outbound Shipment Confirmation	F4201Z1, F4211Z1, F49211Z1	N/A	N/A		P4211Z1	R4211Z1P	P4205
Logistics							
Cycle Counts	F4141Z1	R4141Z1I	R4141Z1I – ZJDE0001	N/A	P4141Z1	R4141Z1P	N/A
Item Master	F4101Z1, F4101Z1A	R4101Z1I	R4101Z1I – ZJDE0001		P4101Z1	R4101Z1P	P4101
Item Cost	F4105Z1	N/A	R4105Z1I – XJDE0001		P4105Z1	R4105Z1P	P4105
Warehouse Confirmations (Suggestions)	F4611Z1	R4611Z1I	R4611Z1I – ZJDE0001		P4611Z1	R4611Z1P	N/A

Interoperability

Manufacturing Work Order Header	F4801Z1	Use Work Order Comple- tions	Use Work Order Comple- tions	R4101Z1O	P4801Z1	R4801Z1P	P48013
Work Order Parts List	F3111Z1	Use Planning Messages	Use Planning Messages		P4801Z1	R3111Z1P	P3111
Work Order Routing	F3112Z1	Use Planning Messages	Use Planning Messages	R4801Z2X	P4801Z1	R3112Z1P	P3112
Work Order Employee Time Entry	F31122Z1	R31122Z1I - XJDE0002	R31122Z1I - XJDE0001		P31122Z1	R31122Z1	P311221
Work Order Inventory Issues	F3111Z1	R31113Z1I - ZJDE0002	R31113Z1I - ZJDE0001		P3111Z1	R3111Z1P	N/A
Work Order Completions	F4801Z1	R31114Z1I - XJDE0002	R31114Z1I - XJDE0001		P4801Z1	R4801Z1P	N/A
Super Backflush	F3112Z1	R31123Z1I	R31123Z1I - ZJDE0001		P3112Z1	R3112Z1P	N/A
Bill of Material	F3002Z1	R3002Z1I - ZJDE0002	R3002Z1I - ZJDE0001		P3002Z1	R3002Z1P	P3002
Routing Master	F3003Z1	R3003Z1I - ZJDE0002	R3003Z1I - ZJDE0001		P3003Z1	R3003Z1P	P3003
Work Center Master	F30006Z1	R30006Z1I - ZJDE0002	R30006Z1I - ZJDE0001		P30006Z1	R30006Z1P	P3006
Work Day Calendar	F0007Z1	R0007Z1I - XJDE0002	R0007Z1I - XJDE0001		P0007Z1	R0007Z1P	P00071
Planning Messages	F3411Z1	R3411Z1I - ZJDE0002	R3411Z1I - ZJDE0001		P3411Z1	R3411Z1P	N/A
Detail Forecast	F3460Z1	R3460Z1I - XJDE0002	R3460Z1I - XJDE0001		P3460Z1	R3460Z1P	P3460, R3465, R34650 (Each clone individually)
Kanban Transactions	F30161Z1	R30161Z1I - XJDE0002	R30161Z1I - XJDE0001	N/A	P30161Z1	R30161Z1P	N/A



Appendix C - Business Function Documentation

Business function documentation explains what individual business functions do and how they should be used. This section includes information to help you generate documentation for business functions. You can generate information for all business functions, groups of business function, or individual business functions.



Business Function Documentation

Business function documentation explains what individual business functions do and how they should be used. The documentation for a business function should include information such as:

- Purpose
- Parameters (the data structure used)
- Explanation of each individual parameter that indicates: input/output required and explanation of return values
- Related tables (the table accessed)
- Related business functions (business functions called from within the functions itself)
- Special handling instructions

You use Business Function Design and Data Structure Design to document your business functions.

Business function documentation features allow you to:

- Create documentation
- Generate documentation
- View documentation

Creating Business Function Documentation

You can create business function documentation at several levels, including:

Business Function Notes This shows you the documentation for the specific business function you are using.

Data Structure Notes This displays notes on the data structure for the business function.

Parameter Notes This displays notes on the actual parameters in the data structure.

Creating Business Function Documentation

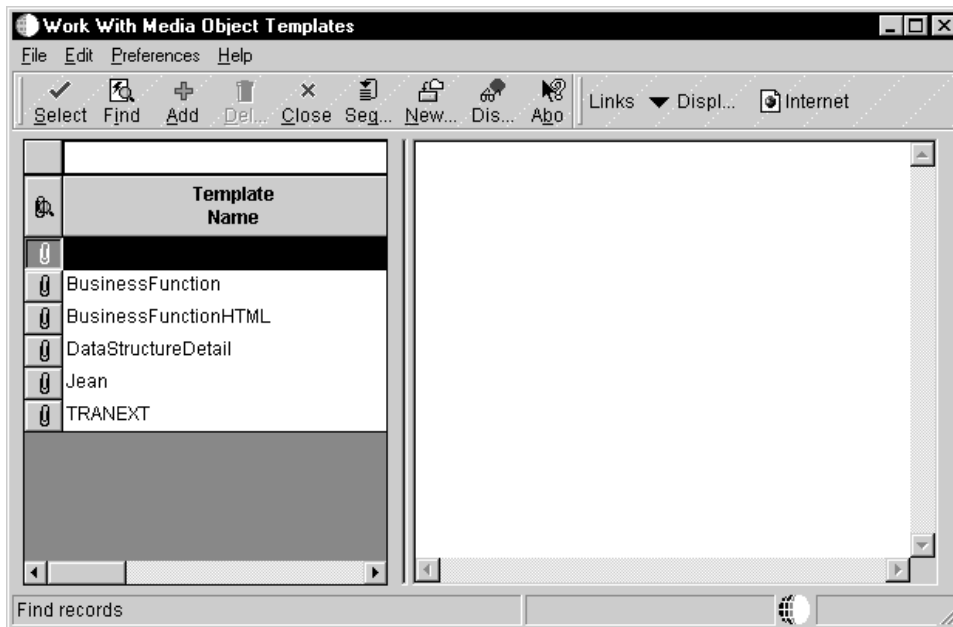
► **To create business function documentation**

1. On Object Management Workbench, choose the business function you wish to document and click the Design button.

The Object Librarian Business Function Design form appears.

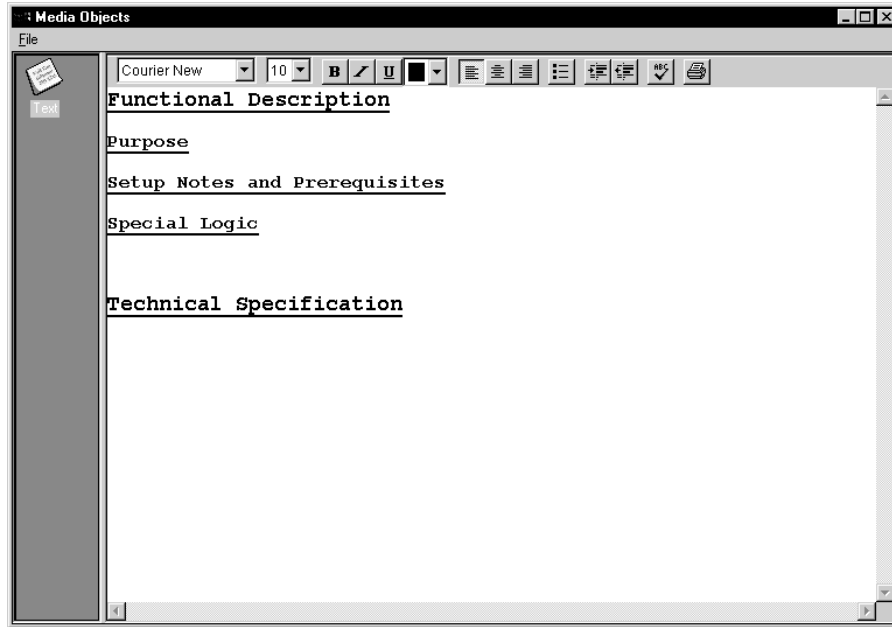
2. Click the Attachments tab.
3. On Media Objects, right click in the icon panel and choose Templates.

Work with Media Object Templates displays the available templates you can use.



4. Select the template you wish to use.

J.D. Edwards uses the Business Function template as a standard.



5. Type in the appropriate information under each template heading.

Business Function Documentation Template

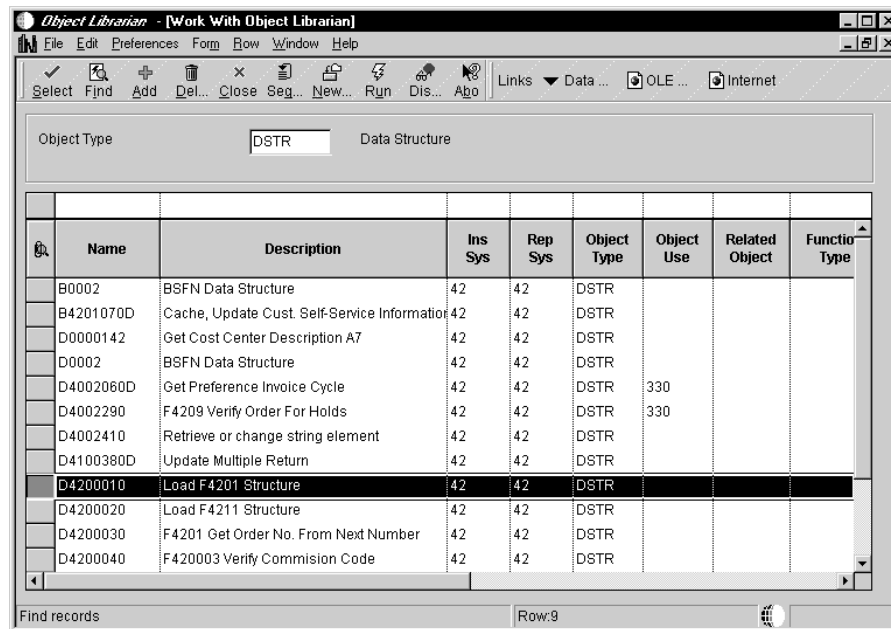
The Business Function documentation template contains the following sections:

Purpose	This section contains a brief summary of what the function does.
Setup Notes and Prerequisites	The section includes any special notes to assist in using the function, including prerequisite functions, special values that need to be initialized, events recommended to run the function, or if memory must be cleared separately after the function is used.
Special Logic	This section contains additional details about the business function logic. It is usually only used for complex functions that require more explanation than the purpose summary.
Technical Specification	This section contains the technical specification of the function written in scripted English. This may be a direct copy from an existing word processing document.

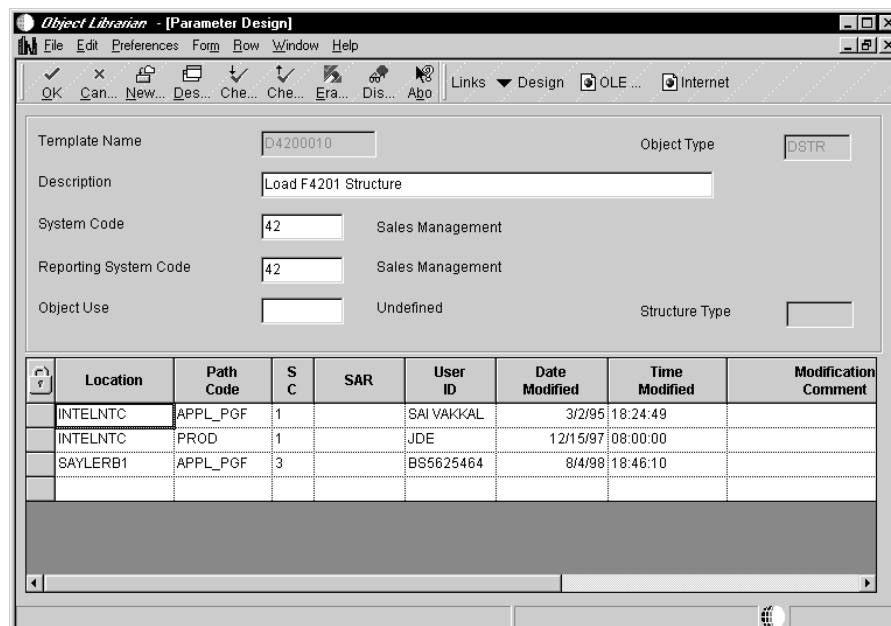
Creating Data Structure Documentation

► **To create data structure documentation**

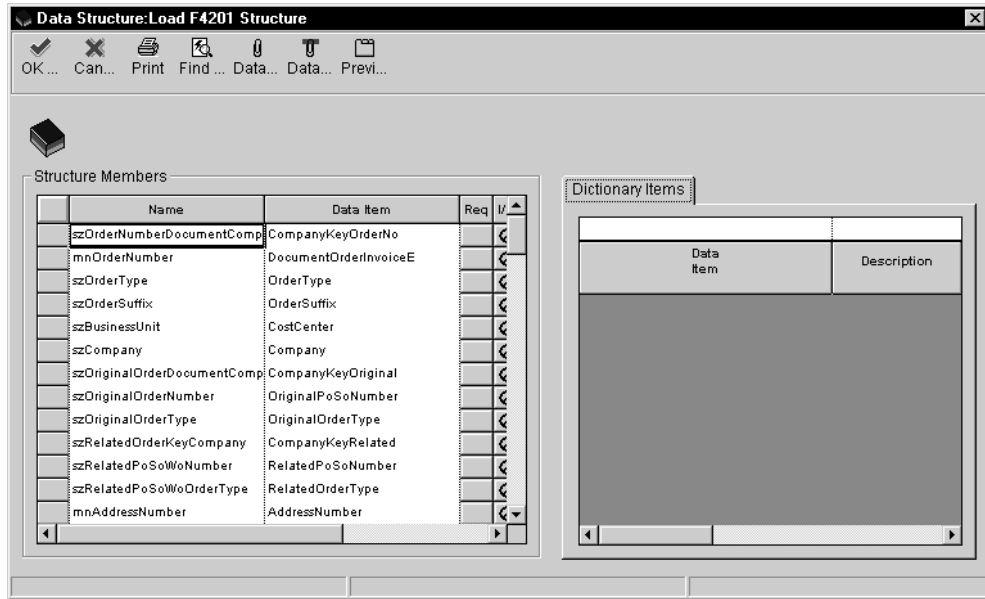
1. On Object Librarian, choose the data structure you wish to document.



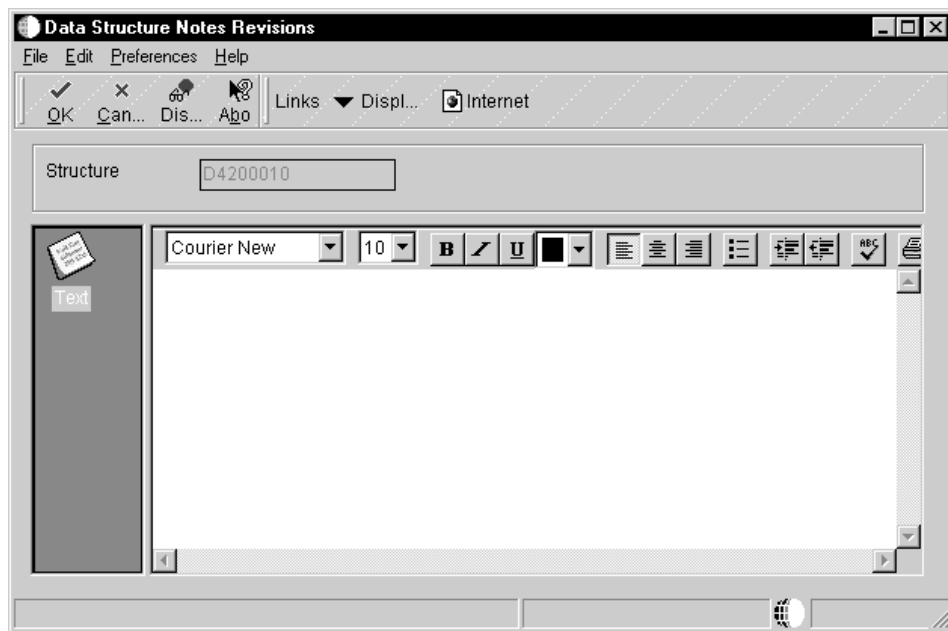
2. Check the data structure out to your workstation.



3. On Parameter Design, choose Design from the Form menu.

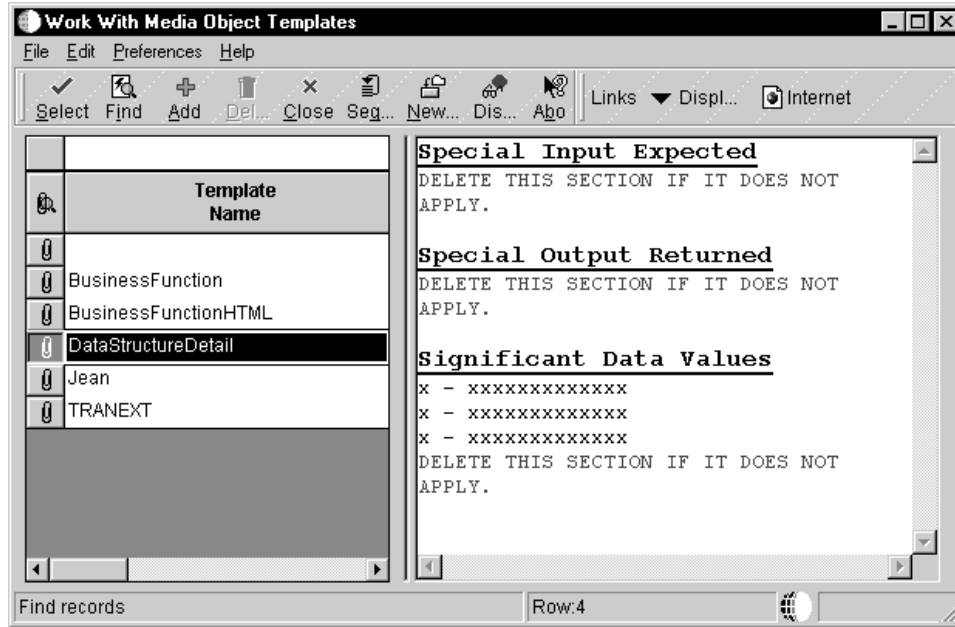


- On Data Structure Design, click the left binder clip for Data Structure Attachments.



- On Media Objects, right click in the icon panel and choose templates.

Work with Media Object Templates displays the available templates you can use.



6. Select the template you wish to use.
 J. D. Edwards uses the Data Structure Detail template.
7. Type in the appropriate information under each template heading.

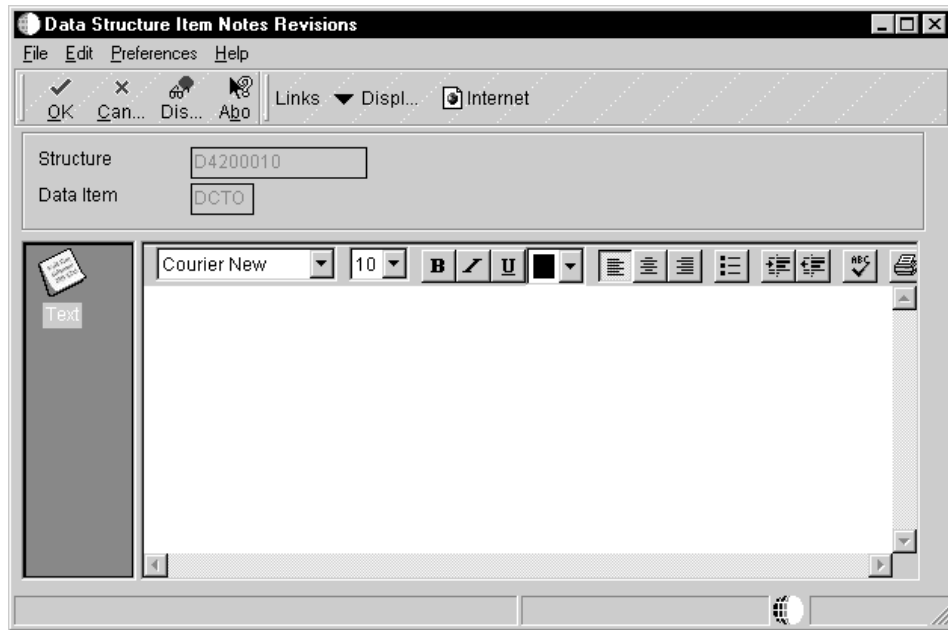
Data Structure Documentation Template

The data structure documentation template contains the following sections:

- Special Input Expected** This section should be deleted if it does not apply.
- Special Output Returned** This section should be deleted if it does not apply.
- Significant Data Values** This section should be deleted if it does not apply. Otherwise it is in the format: x - xxxxxxxxxxxxxxxx.

Creating Parameter Documentation

The steps for creating parameter documentation are the same as those for data structure documentation, except that after selecting the data item in the structure you wish to enter notes for, you click the Data Structure Item Attachments binder clip instead of the Data Structure Attachments binder clip. There is no special template for parameter documentation.



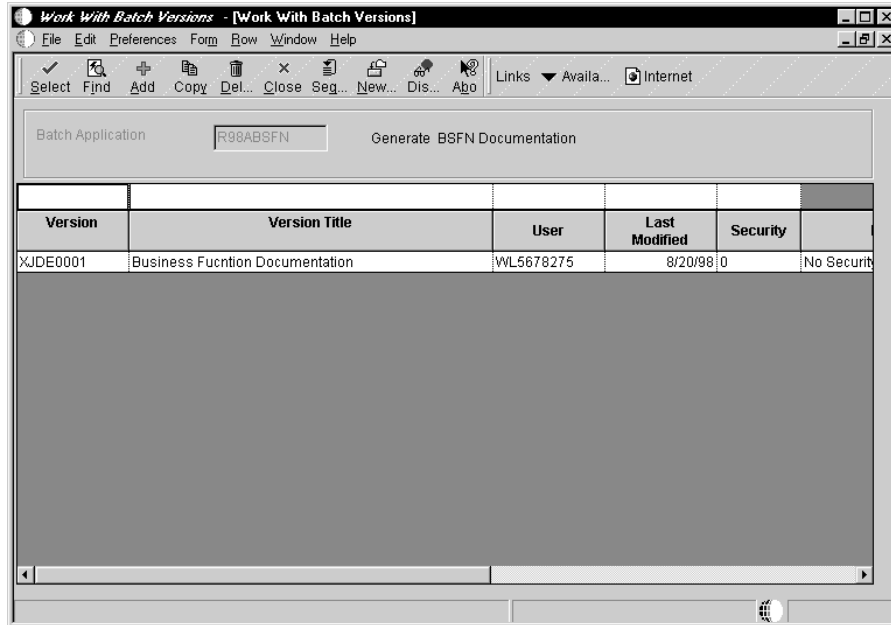
You can enter specific notes about a data structure item to further clarify the information that should be passed in or passed out of the item, for example a mode parameter. The notes should indicate the valid values the function will accept when you hook it up and how to use them. For example, 1 = Add mode, or 2 = Delete.

Generating Business Function Documentation

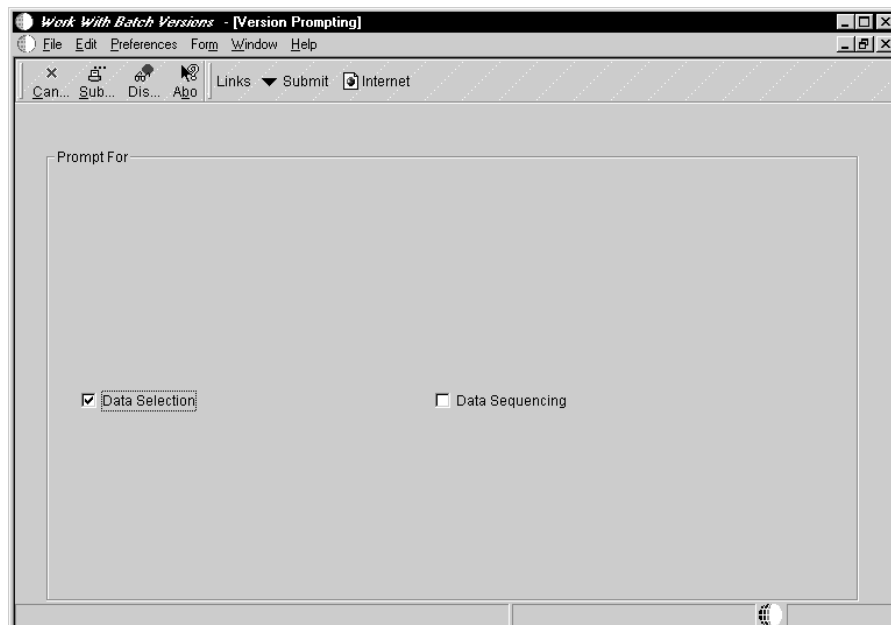
Generating business function documentation provides you with an online list of business function documentation that allows you to view documentation through the Business Function Documentation Viewer (P98ABSFN). Typically the system administrator performs this task, because generating the business function documentation for all business functions takes a long time. If you create new business function documentation you may need to regenerate the business function documentation just for that business function.

► To generate business function documentation

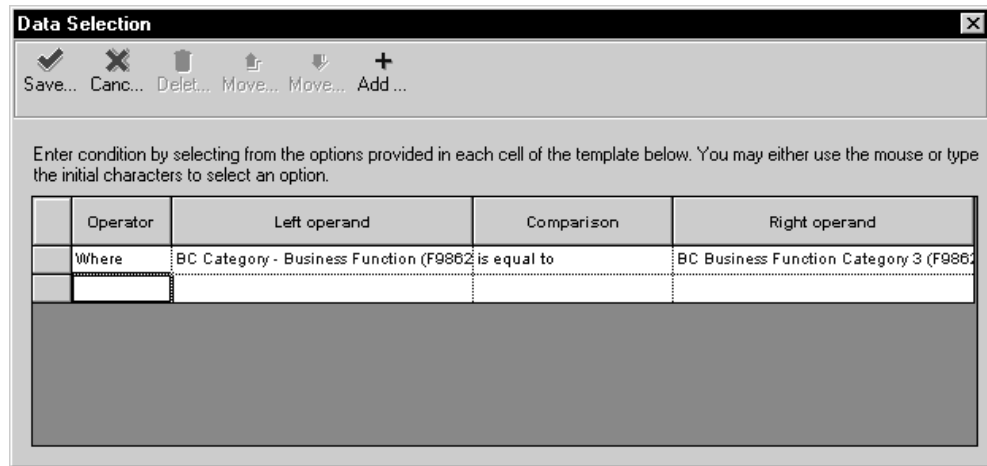
From the Cross Application Development Tools menu (GH902), select Generate BSFN Documentation.



1. On Work with Batch Versions, choose version XJDE0001.



2. If you do not want to generate all business function documentation, on Version Prompting, choose the following option:
 - Data Selection
3. Click Submit.

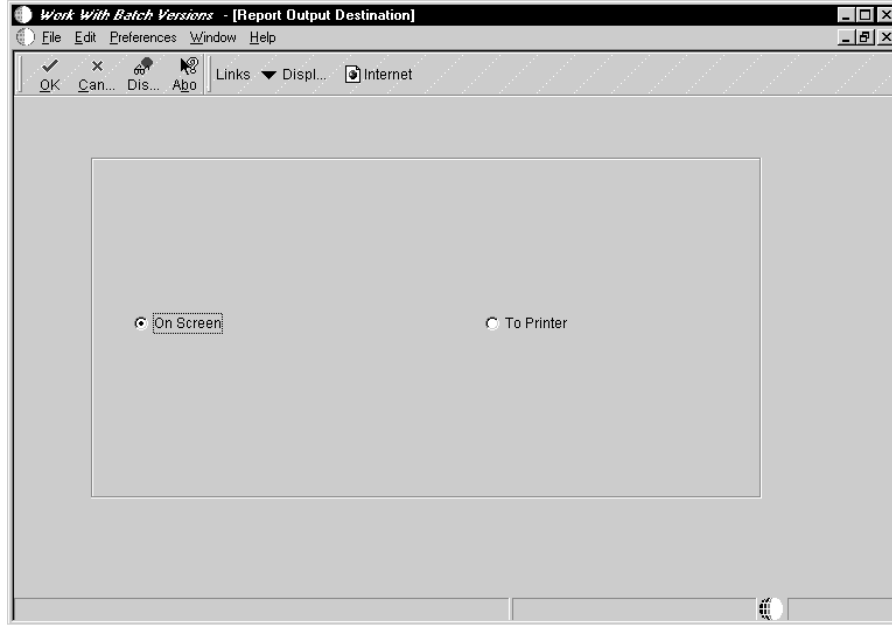


4. On Data Selection, build your criteria for data selection and click OK.

Select only those functions for which you are generating documentation.



5. Depending on the criteria you choose, you might also need to designate processing options.



6. If you are running your report locally, on Report Output Destination, choose one of the following output destinations:
 - On Screen
 - To Printer

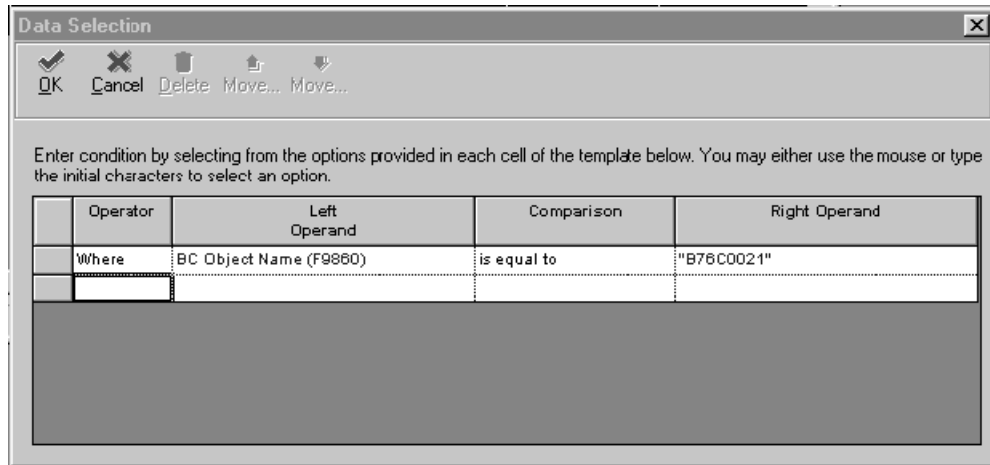
A hypertext markup language (HTML) link is created for each business function for which you generated documentation. An Index HTML file is also created. These HTML files are placed in your output queue directory. Output is in the following format:

Function Name				
<i>Function Description from O/L</i>				
Parent DLL:				
Location:				
Language:				
Purpose				
Special Handling				
Data Structure				
Parameter Name	dataitem	data type	req/opt	i/o/both

Data Selection Tips

You can use data selection to choose the business functions for which you wish to generate documentation. R98ABSFN uses your data selection criteria to filter the business function documentation. It takes longer to run when you generate documentation you do not need. If you generate documentation for all business functions, the process can take quite a while. You can use data selection to generate documentation for one business function, all business functions, or any combination between.

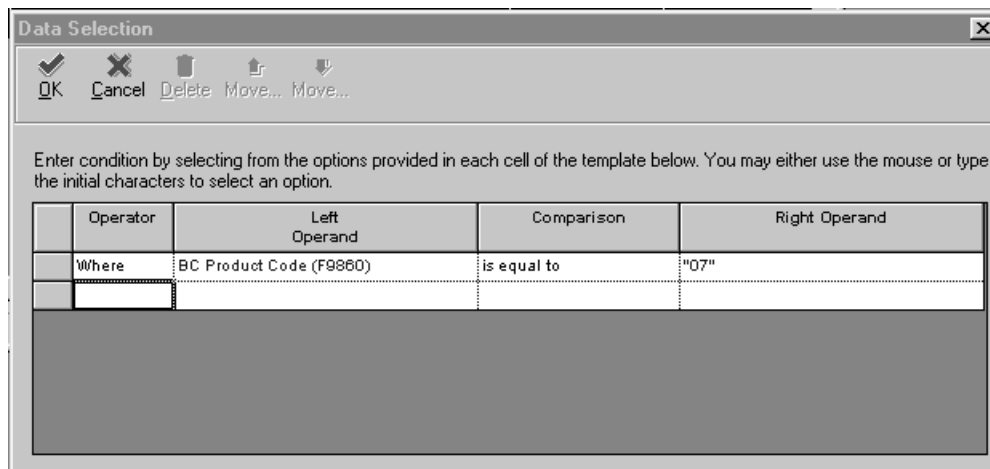
For example, if you want to generate documentation for a single business function you can use the data item BC Object Name (F9860).



The screenshot shows a dialog box titled "Data Selection" with a close button (X) in the top right corner. Below the title bar is a toolbar with icons for OK (checkmark), Cancel (X), Delete (trash), Move... (up arrow), and Move... (down arrow). Below the toolbar is a text area with the instruction: "Enter condition by selecting from the options provided in each cell of the template below. You may either use the mouse or type the initial characters to select an option." Below the text area is a table with four columns: Operator, Left Operand, Comparison, and Right Operand. The first row contains the following values: "Where", "BC Object Name (F9860)", "is equal to", and "'B76C0021'". The second row is empty, and the third row is a shaded area.

	Operator	Left Operand	Comparison	Right Operand
	Where	BC Object Name (F9860)	is equal to	"B76C0021"

If you want to generate documentation for all of the business functions for a specific product code, such as Payroll, you use the data item BC Product Code (F9860).



The screenshot shows a dialog box titled "Data Selection" with a close button (X) in the top right corner. Below the title bar is a toolbar with icons for OK (checkmark), Cancel (X), Delete (trash), Move... (up arrow), and Move... (down arrow). Below the toolbar is a text area with the instruction: "Enter condition by selecting from the options provided in each cell of the template below. You may either use the mouse or type the initial characters to select an option." Below the text area is a table with four columns: Operator, Left Operand, Comparison, and Right Operand. The first row contains the following values: "Where", "BC Product Code (F9860)", "is equal to", and "'07'". The second row is empty, and the third row is a shaded area.

	Operator	Left Operand	Comparison	Right Operand
	Where	BC Product Code (F9860)	is equal to	"07"

You can also use the right operand on the Data Selection form to choose ranges or lists of values to further refine your filter.

You can filter using any value that is associated with a business function. For example, you can use BC Date – Updated (F9860) if you have already produced the documentation for a previous release of OneWorld and you want only new or modified business function documentation after an upgrade or update of OneWorld.

You use BC Function Type (F9860) to choose Master business function documentation.

You use BC Location Business Function (F9860) to produce documentation for client run business functions.

You use BC Object Type (F9860) to generate documentation for NERs only.

You can use many other informational fields to choose the business functions for which you wish to generate documentation.

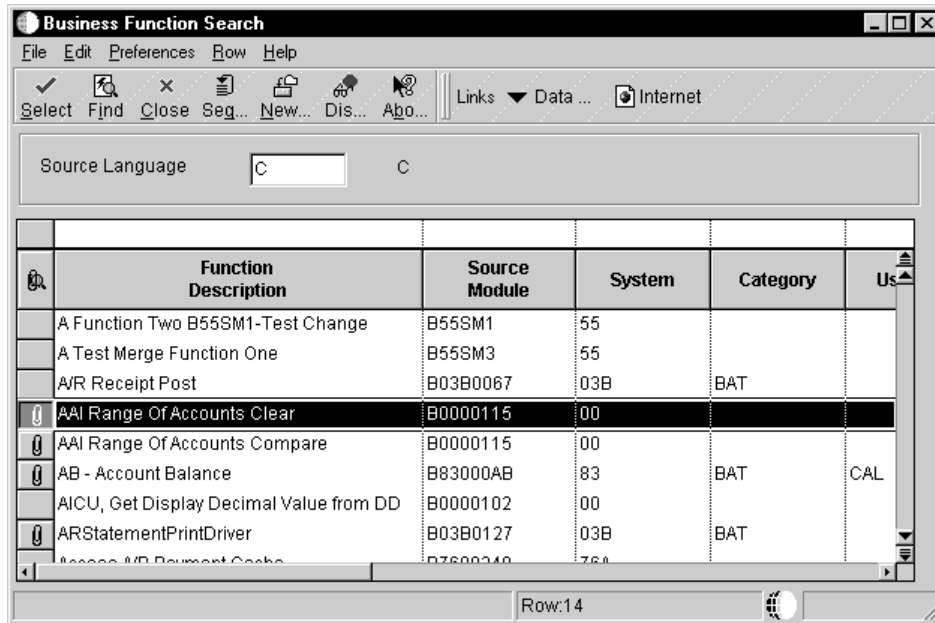
Viewing Business Function Documentation

You can view your business function notes from several different locations, including:

- Business Function Search
- Business Function - Values to Pass
- Business Function Documentation Viewer

Viewing Documentation from Business Function Search

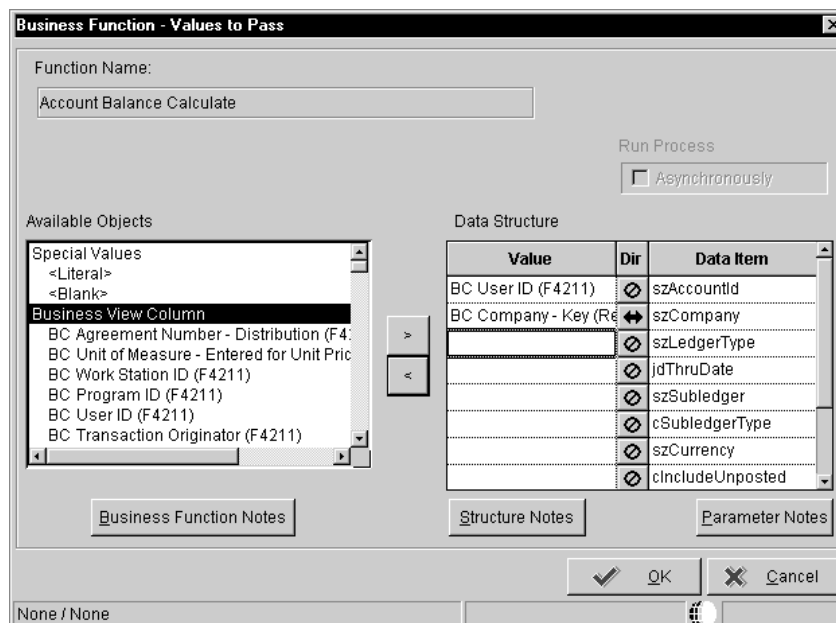
When you make a connection to a business function in event rules, the Business Function Search form appears. You can then select the function you want to call. From the row menu, choose Data Structure Notes or Attachments to view the documentation for the business function.



Viewing Documentation from Business Function - Values to Pass

You can click one of the following buttons on Business Function - Values to Pass to view documentation for a single business function (refer to *Business Function Event Rules* for more information about accessing this form).

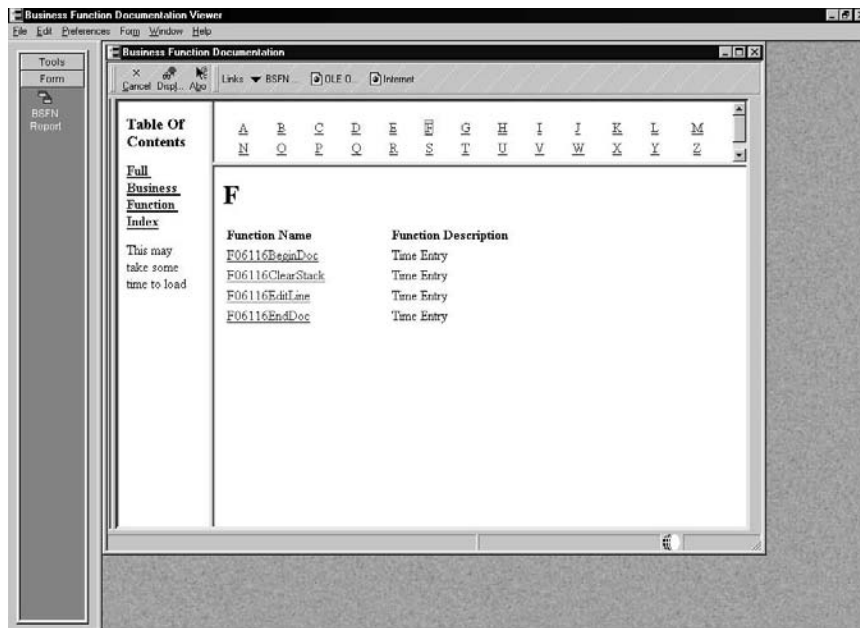
- Business Function Notes
- Structure Notes
- Parameter Notes



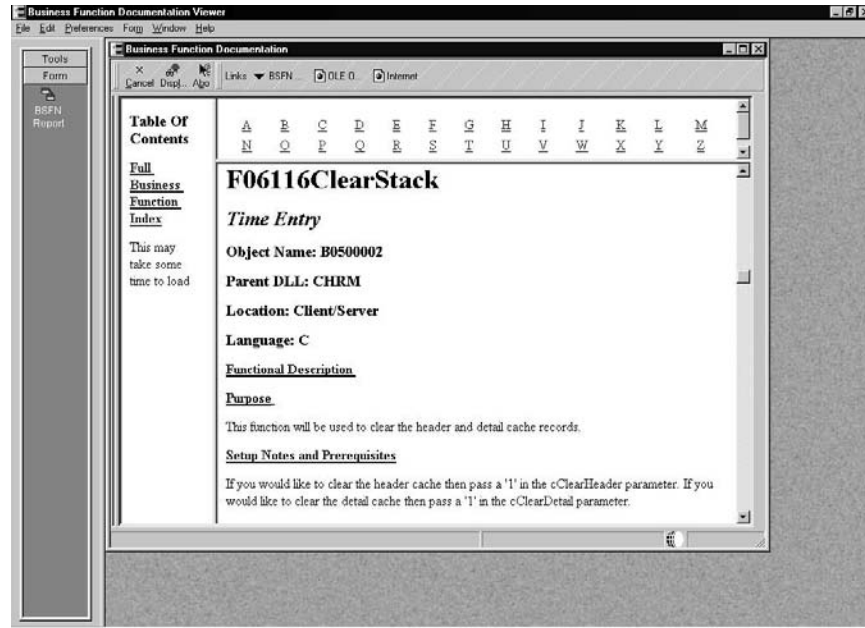
- BSFN Notes** Displays the notes for the business function.
- Structure Notes** Displays the notes for the whole data structure.
- Parameter Notes** Displays the notes for a particular parameter.

Viewing Documentation from Business Function Documentation Viewer

You can use Business Function Documentation Viewer to view documentation for all business functions or selected business functions. Once you have generated your report, use menu GH902 to access the Business Function Documentation Viewer (P98ABSFN) to display your information. J.D. Edwards suggests that you use this method to view business function documentation.



The Business Function Documentation form contains the HTML index you generated. You can view either the entire index or select just the functions for a specific letter in the alphabet by clicking on that letter in the index. Double-click a business function to view documentation specific to that function.



The media object loads the HTML index of the business functions based on a media object queue. In the media object queue table a queue named Business Function Doc must be set up. This queue must point to the directory where the business function HTMLs are located. The system administrator usually generates the documentation for all business functions. Because the generation process places the documentation files in the local directory, the administrator must then copy the files to a central directory on the deployment server. The files must be copied to the media object queue for media object business function notes. If you are running standalone, this path will usually be the output directory from the Network Queue Settings section of our jde.ini file. If this entry is not in your jde.ini file, it is the print queue directory in your OneWorld directory.



Appendix D - Open Data Access (ODA)

The J.D. Edwards OneWorld Open Data Access ODBC driver (ODA) is a version 2.5 or higher compliant, read-only driver. ODA can be used by front-end Windows Query and Reporting tools, such as Microsoft Query or Microsoft Access, to access the J.D. Edwards OneWorld database. ODA sits between the front-end Query/Reporting tools and the OneWorld-configured ODBC drivers.

The J.D. Edwards OneWorld database contains object and column names, specific data types and security rules that must be converted or applied so that the data is presented correctly. The specific data types and rules include Decimal Shifting, Julian Date, Currency, Security, and user defined codes. In some instances, ODA modifies the SQL SELECT statement, as well as the data, so that it appears correctly within the selected tool.

This section includes information about:

- Adding an ODA data source
- Working with ODA
- ODA Error Messages

Hardware and Software Requirements

To use the J.D. Edwards OneWorld Open Data Access Driver you must meet several hardware and software requirements.

Hardware Requirements

To use the J.D. Edwards OneWorld Open Data Access Driver, you must have:

- An IBM-compatible personal computer
- A hard disk with 6 MB of free disk space.
- At least 16 MB of random access memory (RAM).



Software Requirements

To access data with the J.D. Edwards OneWorld Open Data Access driver, you must have:

- J.D. Edwards OneWorld version B732 or later.
- The J.D. Edwards OneWorld Open Data Access driver (JDEOWODA.dll).
- The 32-bit ODBC Driver Manager, version 3.0 or later (ODBC32.dll). Note that this file is included with the ODBC Database Drivers.
- Microsoft Windows 95 or later, or Windows NT 4.0 or later.

The use of this ODBC driver by 16-bit applications on Windows 95 is not supported.

ODBC Component Files

The J.D. Edwards OneWorld installation installs the components required by ODBC Database Drivers. You may also find the following additional files.

Driver	File Name
ODA Driver	JDEOWODA.DLL
ODA Driver Help	JDEOWODA.HLP
Release Notes	README.TXT

Open Data Access Driver Architecture

The J.D. Edwards Open Data Access ODBC driver architecture has five components:

- Application – Front-end Query/Reporting tool that calls the JDE ODBC driver to access data from the JDE database.
- Manager – Load and unloads drivers on behalf of an application. Processes ODBC calls or passes them to the driver.
- J.D. Edwards OneWorld Open Data Access Driver – Passes some of the ODBC requests directly to the vendor's ODBC driver. If J.D. Edwards OneWorld specific data types are used then the SQL SELECT statement is modified before sending it to the vendor's ODBC driver. After the data is returned from the vendor's ODBC driver the J.D. Edwards Open Data

Access ODBC driver may need to manipulate the data so that it is displayed correctly in the application.

- Vendor Driver – Processes ODBC function calls and submits SQL requests to the specific data source. If necessary, the driver modifies an application's request so that the request conforms to the syntax supported by the associated DBMS.
- Data Source – Consists of the data the user wants to access and its associated operating system, DBMS, and network platform.

Adding an ODA Data Source

Although the ODA driver is automatically registered as part of the OneWorld installation process, you may need to add a data source. You can also add a file data source or a system data source, modify a data source, or delete a data source if needed. You might also need to check the currency value check box when you configure the OneWorld ODA driver so that you can view currency data in the correct format.

▶ Adding a Data Source

1. Double-click the Control Panel icon. On Control Panel, double-click the ODBC icon.
2. On User Data Sources dialog box, click Add.
3. On Add Data Source, select the J.D. Edwards OneWorld Open Data Access driver from the Installed ODBC Drivers list and click Finish.
4. On Configure Data Source, enter the following information to set up the data source and click OK.
 - Data Source Name - specify the name that you want to call the J.D. Edwards OneWorld Open Data Access driver. (You must add a driver for each J.D. Edwards OneWorld database that you need to access).
 - Description - specify the description of the driver that you are adding. (Note the Description entry cannot exceed 79 characters.)
5. On the Connect form, enable one or more of the following options:
 - Convert User Defined Codes - Use this option to return the associated description of the user defined field instead of the user defined code. The associated description is more descriptive, because it is a text description instead of a code that is used for the user defined code. The default is to display the associated description instead of the user defined code.
 - Convert Currency Values - Use this option to convert currency fields to the correct values.
 - Use Long Table/Business View Names - Use this option to view long table/view names.
 - Use Long Column Names - Use this option to view long column names.

6. On the Connect form, enable one or more of the following Table/Business View Display Options:
 - Tables Only - Use this option to view only J.D. Edwards OneWorld tables.
 - Business Views Only - Use this option to view only J.D. Edwards OneWorld business views.
 - Tables and Business Views - Use this option to view both J.D. Edwards OneWorld tables and J.D. Edwards OneWorld business views.

Adding a File Data Source

You can also add a file data source.

To add a file data source

1. Double-click the Control Panel icon. On Control Panel, double-click the ODBC icon.
2. On User Data Sources click the DSN tab.
3. On File Data Sources, click Add.
4. On Add Data Source, select the J.D. Edwards OneWorld Open Data Access driver from the Installed ODBC Drivers list and click Finish.
5. On Configure Data Source, enter the following information to set up the data source and click OK.
 - Data Source Name - specify the name that you want to call the J.D. Edwards OneWorld Open Data Access driver. (You must add a driver for each J.D. Edwards OneWorld database that you need to access).
 - Description - specify the description of the driver that you are adding. (Note the Description entry cannot exceed 79 characters.)
6. On the Connect form, enable one or more of the following options:
 - Convert User Defined Codes - Use this option to return the associated description of the user defined field instead of the user defined code. The associated description is more descriptive, because it is a text description instead of a code that is used for the user defined code. The default is to display the associated description instead of the user defined code.
 - Convert Currency Values - Use this option to convert currency fields to the correct values.
 - Use Long Table/Business View Names - Use this option to view long table/view names.

-
- Use Long Column Names - Use this option to view long column names.
7. On the Connect form, enable one or more of the following Table/Business View Display Options:
 - Tables Only - Use this option to view only J.D. Edwards OneWorld tables.
 - Business Views Only - Use this option to view only J.D. Edwards OneWorld business views.
 - Tables and Business Views - Use this option to view both J.D. Edwards OneWorld tables and J.D. Edwards OneWorld business views.

Adding a System Data Source

A data source can be set up with a system data source name (DSN) that can be used by more than one user on the same machine. The system DSN can also be used by a systemwide service, which can then gain access to the data source even if no user is logged onto the machine.

► To add a system data source

1. Double-click the Control Panel icon. On Control Panel, double-click the ODBC icon.
2. On Data Sources, click the System DSN tab, and then click Add.
3. On System Data Sources, click Add.
4. On Add Data Source, select the J.D. Edwards OneWorld Open Data Access driver from the Installed ODBC Drivers list and click Finish.
5. On Configure Data Source, enter the following information to set up the data source and click OK.
 - Data Source Name - specify the name that you want to call the J.D. Edwards OneWorld Open Data Access driver. (You must add a driver for each J.D. Edwards OneWorld database that you need to access).
 - Description - specify the description of the driver that you are adding. (Note the Description entry cannot exceed 79 characters.)
6. On the Connect form, enable one or more of the following options:
 - Convert User Defined Codes - Use this option to return the associated description of the user defined field instead of the user defined code. The associated description is more descriptive, because it is a text description instead of a code that is used for the user defined code. The default is to display the associated description instead of the user defined code.

- Convert Currency Values - Use this option to convert currency fields to the correct values.
 - Use Long Table/Business View Names - Use this option to view long table/view names.
 - Use Long Column Names - Use this option to view long column names.
7. On the Connect form, enable one or more of the following Table/Business View Display Options
- Tables Only - Use this option to view only J.D. Edwards OneWorld tables.
 - Business Views Only - Use this option to view only J.D. Edwards OneWorld business views.
 - Tables and Business Views - Use this option to view both J.D. Edwards OneWorld tables and J.D. Edwards OneWorld business views.

Modifying a Data Source

You can also modify a data source.

To modify a data source

1. Double-click the Control Panel icon. On Control Panel, double-click the ODBC icon.
2. On User Data Sources, File Data Sources, or System Data Sources, choose a data source from the available list.
3. Click Configure.
4. On Configure Data Source, enter the following information to set up the data source and click OK.
 - Data Source Name - specify the name that you want to call the J.D. Edwards OneWorld Open Data Access driver. (You must add a driver for each J.D. Edwards OneWorld database that you need to access).
 - Description - specify the description of the driver that you are adding. (Note the Description entry cannot exceed 79 characters.)
5. On the Connect form, enable one or more of the following options:
 - Convert User Defined Codes - Use this option to return the associated description of the user defined field instead of the user defined code. The associated description is more descriptive, because it is a text description instead of a code that is used for the user defined code. The default is to display the associated description instead of the user defined code.

-
- Convert Currency Values - Use this option to convert currency fields to the correct values.
 - Use Long Table/Business View Names - Use this option to view long table/view names.
 - Use Long Column Names - Use this option to view long column names.
6. On the Connect form, enable one or more of the following Table/Business View Display Options:
- Tables Only - Use this option to view only J.D. Edwards OneWorld tables.
 - Business Views Only - Use this option to view only J.D. Edwards OneWorld business views.
 - Tables and Business Views - Use this option to view both J.D. Edwards OneWorld tables and J.D. Edwards OneWorld business views.

Deleting a Data Source

You can delete a data source.

To delete a data source

1. Double-click the Control Panel icon. On Control Panel, double-click the ODBC icon.
2. On User Data Sources, File Data Sources, or System Data Sources, choose the data source you want to delete from the Data Sources list.
3. Click Remove, and then click Yes to confirm the deletion.

Working with ODA

Once the ODA driver is properly installed and an ODBC data source is established, you can use the ODA driver's functionality. When a SQL connection is established, the environment of the current connection is stored in the system as the database name. This value can later be accessed by SQLGetInfo or it can be used for future connections.

There are several features you can use with J.D. Edwards ODA that are specific to J.D. Edwards.

Long Names

Long Table and Business View Names

Long table and business view names allow you to see a descriptive name when you view an object list. You can use either the descriptive names or the original J.D. Edwards OneWorld object name in the SELECT statement.

Note: This option may not be available for all third-party products, for example, ShowCase STRATEGY products prior to the 2.0 release or Crystal Reports, because the long names contain special characters that are not handled correctly by these tools.

Long Column Names

Long column names allow you to see a descriptive name when viewing any columns list. You can still use either the descriptive names or the original J.D. Edwards OneWorld column name. For example, you can use either of the following statements to retrieve information from the Address Book Master table (F0101):

- SELECT ABAN8 from the Address Book Master table (F0101)
- SELECT AddressNumber from the Address Book Master table (F0101)

Julian Date

Julian Date modifies all references to Julian date columns to convert the date to an SQL-92 standard date. The J.D. Edwards Julian date is converted to a standard date value that can be used in date calculations. This allows you to use a duration or other date calculations in both the select (result data), where and having clauses, and order by.

The SQL SELECT statement is modified to before a data calculation to convert the J.D. Edwards Julian date column to a standard date. The modification to the SQL SELECT statement is based upon the data source that is being accessed because of driver differences in handling date calculations. If the original column value is zero, the date conversion will result in a date value of “1899-12-31”. To remove these values, the following condition should be added to the WHERE clause in the SELECT statement where DATECOL is the J.D. Edwards Julian date column:

```
“DATECOL <> {d ‘1899-12-31’}”
```

Decimal Shifting

All references to decimal shifted columns are modified to shift the decimal point to cause the result data to be correct. This allows SQL statements containing complex expressions, aggregates, and filtering to run and return accurate results.

The SQL SELECT statement is modified to divide the column by the appropriate number of decimal places so that the data is returned correctly and to make compare operators work for filtering.

Currency

Currency columns are limited to single column references in the selected columns list. Returned data is converted using the standard J.D. Edwards currency conversion routines. All other references to the currency column in the SQL statement are passed through to the native driver. You must understand how the currency column is used to make effective use of filtering, for example, a Where clause.

Before selected columns are returned, the J.D. Edwards OneWorld Open Data Access driver converts any currency columns to the correct value. Currency columns used in the WHERE or HAVING clause are processed based on the nonconverted currency value. Currency columns in the GROUP BY or ORDER BY clause are grouped and sorted by the nonconverted currency value.

Security

Column Security

When column security is active, any references to restricted columns causes an error to be returned when the SELECT statement is examined. This includes the use of * (asterisk - selecting all columns) in the select clause, as defined by the SQL-92 standards. This means that you will receive an error if you are not authorized to all the columns in the table.

Row Security

When row security is active, the statement is modified to include the appropriate where clause for filtering out secured rows. This means that you will only see rows that you are authorized to access along with getting accurate results using aggregate functions, for example, SUM or AVG.

User Defined Codes

When user defined codes (UDCs) are enabled, you see the associated description instead of the internal code when the column data is returned. This processing affects only the returned data and has no effect on the other parts of the Select statement, for example Where, Order By. This is an optional setting that can be configured when you set up the driver.

Before the UDC is returned to you, the J.D. Edwards OneWorld Open Data Access driver converts the code to the associated description. The UDC columns used in the WHERE or HAVING clause are selected based on the nonconverted code and the UDC columns referenced in the GROUP BY and ORDER BY clause are grouped and sorted by the nonconverted code.

Example

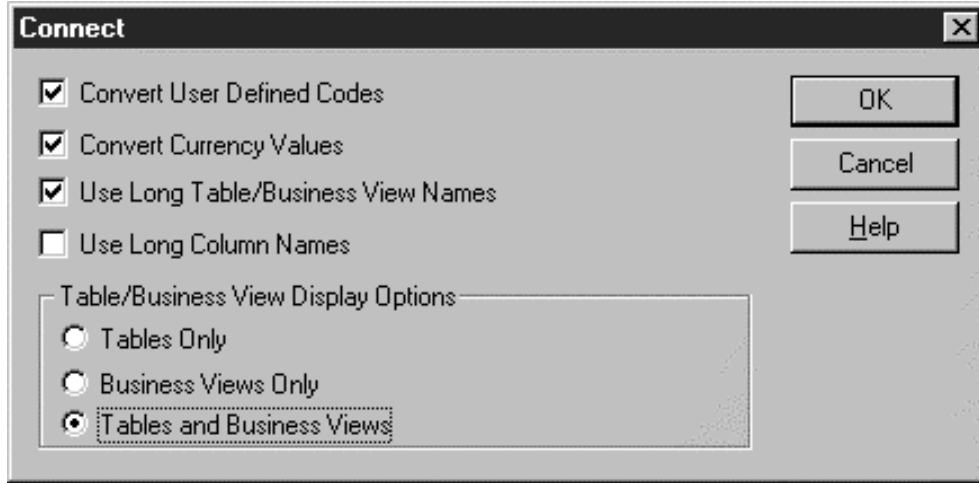
The following example illustrates how you can use Microsoft Excel 97 to create and run a query. To do this complete the following steps:

1. Choose "Get External Data" from the Data menu.
2. Choose "Create New Query"
3. On the Databases tab, choose the appropriate data source (for example, OneWorld Local or OneWorld ODA).

Because Excel uses file data sources, the ODA data source you set up in the 32-bit ODBC Administrator will not appear on the list of databases. You should create a File-type Data Source by selecting <New Data Source> and following the procedures for setting up a data source.

When you choose the ODA data source you may need to log on to OneWorld to use the ODA driver. Once you log on, you will not see OneWorld Explorer because it is only activated so that the ODA driver can check security and environment mappings.

The Excel Query Wizard then displays a list of available tables in the OneWorld data source. Expanding any one table name shows the available columns or fields in each table. If you are using the ODA driver, you will see long descriptions of each field, for example "DateUpdated." If not, you will see the alpha codes for the fields (for example 'ABUPMJ').



- 4 To translate field/column names from the J.D. Edwards alpha codes to something understandable you can use table F9202. Select all rows and sort (on FRDTAI) to create a cross reference. The first two letters of all J.D. Edwards column names are the application code and the remaining letters are in this table as a suffix.
5. Finish building your query with Query Wizard and save the query.

You can then run your query and review it in Excel or MS Query.

After you run a query from Excel, if you ask to see the results in MS Query, you get the results back quickly because it selects a page at a time. If you are working with a large result set, you should close OneWorld and any applications that take up a lot of memory so that you can navigate through the records faster. If you ask for the query to be returned directly to a spreadsheet instead of into MS Query, it may be a long time before you see any rows because Excel does not show a page at a time.

To verify the outcome of each query, you should run each one first using the non-ODA OneWorld data source and then use the ODA data source and compare the results.

ODA Error Messages

Following is a list of the errors that you can receive from the J.D. Edwards Open Data Access driver. The messages are placed in the ODBC error message queue where the application can retrieve them using the standard ODBC error mechanism. The J.D. Edwards messages look like the following:

```
[J.D. Edwards][ODBC J.D. Edwards Driver]MESSAGE TEXT
```

Configuration Request Error

This error may occur when you add a new data source if you do not provide enough information for the driver and it cannot show a configuration dialog.

You must either pass enough information to the driver or allow the driver to prompt for more information.

Option Value Changed

This is an informational message that occurs when you attempt to set a connection or statement option to a value that the driver does not accept. The driver then changes the value to an acceptable default and uses this message to let you know that the value has changed.

The J.D. Edwards Open Data Access driver changes values in the following areas.

- Setting the rowset size to a value other than one. The driver currently only supports single row rowsets.

Data Source Name Is Not Valid

The data source you entered is not a valid ODBC data source name. This error occurs when you are adding a new data source or configuring an existing data source. You must enter a name that follows the ODBC data source naming convention.

Data Source Does Not Exist

This error occurs when you attempt to use a data source that does not exist. You must enter the name of an existing data source. If you get this error when you attempt to connect to a data source, you may need to create a default data source.

Unable to Allocate Memory

The J.D. Edwards OneWorld Open Data Access driver was not able to allocate enough memory to continue. You must close down some open applications and try the operation again. Make sure that you meet the minimum system requirements.

Invalid Type of Request

You attempted to use a configuration option that is unknown to the driver. The driver supports the following options when configuring data sources.

- Adding a data source
- Configuring a data source
- Removing a data source

Data Truncated

The conversion of column data resulted in a truncation of the value. You should allocate more room for the column data to avoid this informational message.

Syntax Error or Access Violation

The statement contained a syntax error and no further information is available.

Unable to Display Connection Dialog

The driver encountered an error when attempting to display the connection dialog.

Cross System Joins Not Supported

This error occurs in one of two situations.

- You referenced tables that are contained on multiple systems in the OneWorld environment. The J.D. Edwards Open Data Access driver currently supports tables referenced on a single system.
- You referenced a business view that contains multiple tables that reside on multiple systems.

You must make sure that you are referencing tables on a single system or a business view that contains tables on a single system.

Unable to Connect to the OneWorld Environment

The driver could not establish a connection to the J.D. Edwards OneWorld environment. This connection is required before a successful connection can be made to this driver.

Internal Data Conversion Error

The driver encountered an unknown error during data conversion.

Internal Execution Error

The driver experienced an unexpected error during a statement execution.

User Defined Code Columns Can Only Be in Simple Column References

A user attempted to use a User Defined Code column in a complex expression. The J.D. Edwards OneWorld Open Data Access driver only allows such columns to be simple references.

Currency Columns Can Only Be in Simple Column References

A user attempted to use a Currency column in a complex expression. The J.D. Edwards OneWorld Open Data Access driver only allows such columns to be simple references.

Column Security Violation

You attempted to use a column you are not authorized to use. You must remove references to those columns that are secured.

Invalid Cursor State

You attempted an operation that was not valid for the state that the driver is in, for example:

- You attempted to bind a column prior to preparing or executing a statement.
- You attempted to execute a statement while there are pending results.
- You attempted to get data from the driver prior to preparing or executing a statement.
- You attempted to prepare a statement while there are pending results.

Invalid Column Number

You attempted to access a column that was not part of the statements results.

Driver Does Not Support the Requested Conversion

An attempt was made to convert a column to a data type that is not supported by the J.D. Edwards OneWorld Open Data Access driver.

Invalid Date/Time String

An attempt to convert a character column to a date, time, or timestamp value failed because the character column did not contain a valid format.

Invalid Numeric String

An attempt to convert a character column to a numeric value failed because the character column did not contain a valid numeric value.

Numeric Value Out of Range

An attempt to convert a column to a numeric value failed because the output data type could not accommodate the value in the column. You should use the default data type or choose a data type that can accommodate the column value.

Data Returned for One or More Columns was Truncated

An attempt to convert a column to a numeric value caused a truncation of decimal digits. The output data type could not accommodate the value in the column. You should use the default data type or choose a data type that can accommodate the column value.

The Data Cannot be Converted

An attempt to convert a column value failed because the input type could not be converted to output type. You should use the default data type.

Statement Must Be a SELECT

The J.D. Edwards OneWorld Open Data Access driver is read only and allows only SELECT statements.

Attempt to Fetch Before the First Row

An attempt was made to fetch before the beginning of results. The attempt resulted in the first rowset being fetched.

Option Value Changed

An attempt was made to set a connection, statement or scroll options to a value that was not allowed. The J.D. Edwards OneWorld Open Data Access driver substituted a similar value.

Fractional Truncation

An attempt to convert a column to a numeric value succeeded with a loss of fractional digits because the output data type could not accommodate the value in the column. You should use the default data type or choose a data type that can accommodate the column value.

Driver Not Capable

An attempt was made to set a connection, statement, or scroll option that the driver does not allow.

Multiple Business Views Referenced

An attempt was made to reference more than one business view in a single SELECT statement. The J.D. Edwards OneWorld Open Data Access driver restricts the SELECT statement to contain only one business view.

Unable to Open Table or Business View

The J.D. Edwards OneWorld Open Data Access driver was unable to locate the table or business view in the OneWorld database or could not get information pertaining to the table or business view.

Server Connection Failed

The J. D. Edwards OneWorld Open Data Access driver was unable to establish a connection to the server referenced by the tables or business view in the SELECT statement.

Business View Contains Invalid Join

The Business View definition contains a join condition that could not be processed by the J. D. Edwards OneWorld Open Data Access driver.

Business View Contains Unsupported UNION Operator

The Business View definition contains the UNION operator that could not be processed by the J. D. Edwards OneWorld Open Data Access driver.



Appendix E – XML Format Examples

This section contains examples of the following XML formats:

- XML Examples (All Parameters)
- XML Examples (Default Parameters)



XML Format Examples (All Parameters)

XML Format Examples (All Parameters)

The following examples are used for specific formats.

Inbound Sales Order XML Format (All Parameters)

This example illustrates an inbound XML format with all of the parameters.

```
    "<?xml version='1.0' ?>
    <jdeRequest type='callmethod' user='userid' pwd='password' environment='environment' >
<callMethod name=' GetLocalComputerId' app='NetCommerce' runOnError='no'>
  <params>
    <param name=' szMachineKey' id='m2' >< /param>
  <params>
<callMethod>
<callMethod name=' F4211FSBeginDoc' app='NetCommerce' runOnError='no'>
  <params>
    <param name='mnCMJobNumber' id='j1'></param>
    <param name='cCMDocAction' >A</param>
    <param name='cCMProcessEdits' <1></param>
    <param name='szCMComputerID' idref='c2' >< /param>
    <param name='cCMErrorConditions' >value< /param>
    <param name='cCMUpdateWriteToWF' >value< /param>
    <param name='szCMProgramID' >value< /param>
    <param name='szCMVersion' >value< /param>
    <param name='szOrderCo' <value< /param>
    <param name='mnOrderNo' >value< /param>
    <param name='szOrderType' >value< /param>
    <param name='szBusinessUnit' >value< /param>
    <param name='szOriginalOrderCo' >value< /param>
    <param name='szOriginalOrderNo' >value< /param>
    <param name='szOriginalOrderType' >value< /param>
```



```
<param name='mnAddressNumber' >value</param>
<param name='mnShipToNo' >value</param>
<param name='jdRequestedDate' >value</param>
<param name='jdOrderDate' >value</param>
<param name='jdPromisedDate' >value</param>
<param name='jdCancelDate' >value</param>
<param name='szReference' >value</param>
<param name='szDeliveryInstructions1' >value</param>
<param name='szDeliveryInstructions2' >value</param>
<param name='szPrintMesg' >value</param>
<param name='szPaymentTerm' >value</param>
<param name='cPaymentInstrument' >value</param>
<param name='szAdjustmentSchedule' >value</param>
<param name='mnTradeDiscount' >value</param>
<param name='szTaxExplanationCode' >value</param>
<param name='szTaxArea' >value</param>
<param name='szCertificate' >value</param>
<param name='cAssociatedText' >value</param>
<param name='szHoldOrdersCode' >value</param>
<param name='cPricePickListYN' >value</param>
<param name='mnInvoiceCopies' >value</param>
<param name='mnBuyerNumber'>value</param>
<param name='mnCarrier' >value</param>
<param name='szRouteCode' >value</param>
<param name='szStopCode' >value</param>
<param name='szZoneNumber' >value</param>
<param name='szFreightHandlingCode' >value</param>
<param name='cApplyFreightYN' >value</param>
<param name='mnCommissionCode1' >value</param>
<param name='mnCommissionRate1' >value</param>
<param name='mnCommissionCode2' >value</param>
```

```
<param name='mnCommissionRate2' >value</param>
<param name='szWeightDisplayUOM' >value</param>
<param name='szVolumeDisplayUOM' >value</param>
<param name='szAuthorizationNo' >value</param>
<param name='szCreditBankAcctNo' >value</param>
<param name='jdCreditBankExpiredDate' >value</param>
<param name='cMode' >value</param>
<param name='szCurrencyCode' >value</param>
<param name='mnExchangeRate' >value</param>
<param name='szOrderedBy' >value</param>
<param name='szOrderTakenBy' >value</param>
<param name='szUserReservedCode' >value</param>
<param name='jdUserReservedDate' >value</param>
<param name='mnUserReservedAmnt' >value</param>
<param name='mnUserReservedNo' >value</param>
<param name='szUserReservedRef' >value</param>
<param name='jdDateUpdated' >value</param>
<param name='szUserID' >value</param>
<param name='szWKBaseCurrency' >value</param>
<param name='cWKAdvancedPricingYN' >value</param>
<param name='szWKCreditMesg' >value</param>
<param name='szWKTempCreditMesg' >value</param>
<param name='cWKInvalidSalesOrderNo' >value</param>
<param name='cWKSourceOfData' >blank</param>
<param name='cWKProcMode' >blank</param>
<param name='mnWKSuppressProcess' >0</param>
<param name='mnSODDocNo' >value</param>
<param name='szSODDocType' >value</param>
<param name='szSODOrderCo' >value</param>
<param name='mnTriangulationRateFrom' >value</param>
<param name='mnTriangulationRateTo' >value</param>
```

```

<param name='cCurrencyConversionMethod' >value</param>
<param name='cRetrieveOrderNo' >value</param>
<param name='szPricingGroup' >value</param>
<param name='cCommitInvInED' >value</param>
<param name='cSpotRateAllowed' >value</param>
<param name='cGenericChar2_EV02' >value</param>
<param name='szGenericString1_DL01' >value</param>
<param name='szGenericString2_DL02' >value</param>
<param name='mnGenericMathNumeric1_MATH01' >value</param>
<param name='mnGenericMathNumeric2_MATH02' >value</param>
<param name='szLongAddressNumberShipto' >value</param>
<param name='szLongAddressNumber' >value</param>
<param name='mnProcessID' >value</param>
<param name='mnTransactionID' >value</param>
</params>
<onError abort='yes'>\
  <callMethod name=' F4211ClearWorkFile' app='NetCommerce' runOnError='yes'>
    <params>
      <param name='mnJobNo' idref='j1'></param>
      <param name='szComputerID' idref='c2' >< /param>
      <param name='mnFromLineNo' >value</param>
      <param name='mnThruLineNo' >value</param>
      <param name='cClearHeaderWF' >value</param>
      <param name='cClearDetailWF' >value</param>
      <param name='szProgramID' >value</param>
      <param name='mnWKRelatedOrderProcess' >value</param>
      <param name='szCMVersion' >value</param>
      <param name='cGenericChar1_EV01' >value</param>
      <param name='szGenericString1_DL01' >value</param>
      <param name='mnSODRelatedJobNumber' >value</param>
      <param name='mnProcessID' >value</param>
    </params>
  </callMethod>
</onError>

```

```
<param name='mnTransactionID' >value</param>
</params>
</callMethod>
</onError>
</callMethod>
<callMethod name=' F4211FSEditLine' app='NetCommerce' runOnError='yes'>
  <params>
    <param name='mnCMJobNo ' idref='j1'></param>
    <param name='cMLineAction' >value</param>
    <param name='cMProcessEdits ' >value</param>
    <param name='cMWriteToWFFlag ' >value</param>
    <param name='cMRecdWrittenToWF ' >value</param>
    <param name='szCMComputerID' idref='c2' >< /param>
    <param name='cMErrorConditions' >value</param>
    <param name='szOrderCo' >value</param>
    <param name='mnOrderNo' >value</param>
    <param name='szOrderType' >value</param>
    <param name='mnLineNo' >value</param>
    <param name='szBusinessUnit' >value</param>
    <param name='mnShipToNo' >value</param>
    <param name='jdRequestedDate' >value</param>
    <param name='jdPromisedDate' >value</param>
    <param name='jdCancelDate' >value</param>
    <param name='jdPromisedDlvryDate' >value</param>
    <param name='szItemNo' >value</param>
    <param name='szLocation'>value</param>
    <param name='szLotNo' >value</param>
    <param name='szDescription1' >value</param>
    <param name='szDescription2' >value</param>
    <param name='szLineType' >value</param>
    <param name='szLastStatus' >value</param>
```

```
<param name='szNextStatus' >value</param>
<param name='mnQtyOrdered' >value</param>
<param name='mnQtyShipped' >value</param>
<param name='mnQtyBackordered' >value</param>
<param name='mnQtyCanceled' >value</param>
<param name='mnExtendedPrice' >value</param>
<param name='mnExtendedCost' >value</param>
<param name='szPrintMesg' >value</param>
<param name='cPaymentInstrument' >value</param>
<param name='szAdjustmentSchedule' >value</param>
<param name='cSalesTaxableYN' >value</param>
<param name='cAssociatedText' >value</param>
<param name='szTransactionUOM' >value</param>
<param name='szPricingUOM' >value</param>
<param name='mnItemWeight' >value</param>
<param name='szWeightUOM' >value</param>
<param name='mnForeignUnitPrice' >value</param>
<param name='mnForeignExtPrice' >value</param>
<param name='mnForeignUnitCost' >value</param>
<param name='mnForeignExtCost' >value</param>
<param name='szPricingCategoryLevel' >value</param>
<param name='mnDiscountFactor' >value</param>
<param name='mnCMLineNo' >value</param>
<param name='szCMProgramID' >value</param>
<param name='szCMVersion' >value</param>
<param name='mnSupplierNo' >value</param>
<param name='szRelatedKitItemNo' >value</param>
<param name='mnKitMasterLineNo' >value</param>
<param name='mnComponentLineNo' >value</param>
<param name='mnRelatedKitComponent' >value</param>
<param name='mnNoOfCpntPerParent' >value</param>
```

```
<param name='cOverridePrice' >value</param>
<param name='cOverrideCost' >value</param>
<param name='szUserID' >value</param>
<param name='jdDateUpdated' >value</param>
<param name='mnWKOrderTotal' >value</param>
<param name='mnWKForeignOrderTotal' >value</param>
<param name='mnWKTotCost' >value</param>
<param name='mnWKForeignTotCost' >value</param>
<param name='cWKProcessingType' >value</param>
<param name='cWKSourceOfData' >value</param>
<param name='cWKCheckAvailability' >value</param>
<param name='mnLastLineNoAssigned' >value</param>
<param name='cStockingType' >value</param>
<param name='szOriginalOrderKeyCo' >value</param>
<param name='szOriginalOrderNo' >value</param>
<param name='szOriginalOrderType' >value</param>
<param name='mnOriginalOrderLineNo' >value</param>
<param name='cParentItmMethdOfPriceCalcn' >value</param>
<param name='szLandedCost' >value</param>
<param name='mnWKSsuppressProcess' >value</param>
<param name='mnShortItemNo' >value</param>
<param name='mnWKRelatedOrderProcess' >value</param>
<param name='mnSODLineNo' >value</param>
<param name='mnPriceAdjRevLevel' >value</param>
<param name='szSalesOrderFlags' >value</param>
<param name='mnSODDocNo' >value</param>
<param name='szSODDocType' >value</param>
<param name='szSODOrderCo' >value</param>
<param name='szTransferOrderToBranch' >value</param>
<param name='mnDomesticDetachedAdj' >value</param>
<param name='mnForeignDetachedAdj' >value</param>
```

```
<param name='mnSODWFLineNo' >value</param>
<param name='szGeneric2CharString' >value</param>
<param name='mnTOEPOExchangeRate' >value</param>
<param name='szTOEPOCurrencyCode' >value</param>
<param name='mnDRPKeyId' >value</param>
<param name='mnSoldToCust' >value</param>
<param name='szF4201BranchPlant' >value</param>
<param name='szSoldToCurrencyCode' >value</param>
<param name='cConsolidationFlag' >value</param>
<param name='jdPriceEffectiveDate' >value</param>
<param name='mnWOWFLineNo' >value</param>
<param name='mnLineNoIncrement' >value</param>
<param name='mnParentWFLineNo' >value</param>
<param name='cStatusInWarehouse' >value</param>
<param name='cBypassCommitments' >value</param>
<param name='szProductSource' >value</param>
<param name='szProductSourceType' >value</param>
<param name='mnSequenceNumber' >value</param>
<param name='szAgreementNumber' >value</param>
<param name='mnAgreementSupplement' >value</param>
<param name='mnAgreementsFound' >value</param>
<param name='szModeOfTransport' >value</param>
<param name='szDutyStatus' >value</param>
<param name='szLineofBusiness' >value</param>
<param name='jdPromisedShip' >value</param>
<param name='szEndUse' >value</param>
<param name='mnTOEPOExchangeRate' >value</param>
<param name='szPriceCode1' >value</param>
<param name='szPriceCode2' >value</param>
<param name='szPriceCode3' >value</param>
<param name='szItemFlashMessage' >value</param>
```

```
<param name='szCompanyKeyRelated' >value</param>
<param name='szRelatedPoSoNumber >value</param>
<param name='szRelatedOrderType >value</param>
<param name='mnRelatedPoSoLineNo>value</param>
<param name='cGenericChar3 >value</param>
<param name='mnProfitMargin >value</param>
<param name='mnQuantityAvailable >value</param>
<param name='cRequestScheduleFlag >value</param>
<param name='cOrderProcessType >value</param>
<param name='cGenericChar2 >value</param>
<param name='mnSODRelatedJobNumber >value</param>
<param name='szGenericString >value</param>
<param name='mnCarrier >value</param>
<param name='szGenericString2_DL02 >value</param>
<param name='mnGenericMathNumeric1_MATH01 >value</param>
<param name='mnGenericMathNumeric2_MATH02 >value</param>
<param name='mnItemVolume_ITVL >value</param>
<param name='szVolumeUOM_VLUM >value</param>
<param name='szRevenueBusinessUnit >value</param>
<param name='szCustomerPO_VR01 >value</param>
<param name='szReference2Vendor_VR02 >value</param>
<param name='mnProcessID >value</param>
<param name='mnTransactionID >value</param>
</params >
<onError abort='no'>\
</onError >
</callMethod >
<callMethod name=' F4211FSEndDoc' app='NetCommerce' runOnError='no'>
<params>
<param name='mnCMJobNo ' idref='j1 '></param>
<param name='mnSalesOrderNo' >value</param>
```



```
<param name='szCMComputerID' idref='c2' >< /param>
<param name='cMErrorCondition' >value</param>
<param name='szOrderType' >value</param>
<param name='szKeyCompany' >value</param>
<param name='mnOrderTotal' >value</param>
<param name='mnForeignOrderTotal' >value</param>
<param name='szBaseCurrencyCode' >value</param>
<param name='szProgramID' >value</param>
<param name='szWorkstationID' >value</param>
<param name='szCMProgramID' >value</param>
<param name='szCMVersion' >value</param>
<param name='mnTimeOfDay' >value</param>
<param name='mnTotalCost' >value</param>
<param name='mnForeignTotalCost' >value</param>
<param name='cSuppressRlvBlnktFlag' >value</param>
<param name='cWKSkipProcOptions' >value</param>
<param name='mnWKRelatedOrderProcess' >value</param>
<param name='cCMUseWorkFiles' >value</param>
<param name='mnEDIDocNo' >value</param>
<param name='szEDIKeyCo' >value</param>
<param name='szEDIDocType' >value</param>
<param name='cCMProcessEdits' >value</param>
<param name='cGenericChar2' >value</param>
<param name='mnSODRelatedJobNumber' >value</param>
<param name='cGenericChar1_EV01' >value</param>
<param name='mnGenericMathNumeric2_MATH02' >value</param>
<param name='szGenericString1_DL01' >value</param>
<param name='szGenericString2_DL02' >value</param>
<param name='mnProcessID' >value</param>
<param name='mnTransactionID' >value</param>
</params/>
```

```

<onError abort='no'>\
  <callMethod name=' F4211ClearWorkFile' app='NetCommerce'
    runOnError='yes'>
    <params>
      <param name='mnJobNo' idref='j1'></param>
      <param name='szComputerID' idref='c2' >< /param>
      <param name='mnFromLineNo' >value</param>
      <param name='mnThruLineNo' >value</param>
      <param name='cClearHeaderWF' >value</param>
      <param name='cClearDetailWF' >value</param>
      <param name='szProgramID' >value</param>
      <param name='mnWKRelatedOrderProcess' >value</param>
      <param name='szCMVersion' >value</param>
      <param name='cGenericChar1_EV01' >value</param>
      <param name='szGenericString1_DL01' >value</param>
      <param name='mnSODRelatedJobNumber' >value</param>
      <param name='mnProcessID' >value</param>
      <param name='mnTransactionID' >value</param>
    </params>
  </callMethod>
</onError>
</callMethod>
<returnParams version='value' messagetype='message' failure Destination='queueName'
successDestination='queueName'>
  <param name='long description' idref='value' /param>
</returnParams>
<onError>
  <callMethod name=' F4211ClearWorkFile' app='NetCommerce'
    runOnError='yes'>
    <params>
      <param name='mnJobNo' idref='j1'></param>

```

```

        <param name='szComputerID' idref='c2' >< /param>
        <param name='mnFromLineNo' >value</param>
        <param name='mnThruLineNo' >value</param>
        <param name='cClearHeaderWF' >value</param>
        <param name='cClearDetailWF' >value</param>
        <param name='szProgramID' >value</param>
        <param name='mnWKRelatedOrderProcess' >value</param>
        <param name='szCMVersion' >value</param>
        <param name='cGenericChar1_EV01' >value</param>
        <param name='szGenericString1_DL01' >value</param>
        <param name='mnSODRelatedJobNumber' >value</param>
        <param name='mnProcessID' >value</param>
        <param name='mnTransactionID' >value</param>
    </params>
</callMethod>
</onError>
</jdeRequest>

```

Outbound Customer Create XML Format (all fields)

This example illustrates an outbound XML format with all of the parameters.

Outbound Customer Create XML Request and Response Format

```

“This format will return all columns for the customer master (F0101Z2)“
<?xml version='1.0' ?>
<jdeRequest type='trans' user='user' pwd='password' environment='environment' session=' ' sessionid='300'>
<transaction action='transactionInfo' type='JDEAB'>
<key>
<column name='EdiUserId'>value</column>
<column name='EdiBatchNumber'>value</column>
<column name='EdiTransactNumber'>value</column>
</key>
</transaction>
</jdeRequest>

```

```
<?xml version='1.0' encoding='utf-8' ?>
<jdeResponse type='trans' user='user' session='session' environment='env'>
  <transaction type='JDEAB' action='transactionInfo'>
    <returnCode code='0'>XML Request OK</returnCode>
    <key>
      <column name='EdiUserId'></column>
      <column name='EdiBatchNumber'></column>
    </key>
    <table name='F0101Z2' type='detail'>
      <column name='EdiUserId'></column>
      <column name='EdiBatchNumber'></column>
      <column name='EdiTransactNumber'></column>
      <column name='EdiLineNumber'></column>
      <column name='EdiDocumentType'></column>
      <column name='TypeTransaction'></column>
      <column name='EdiTranslationFormat'></column>
      <column name='EdiTransmissionDate'></column>
      <column name='DirectionIndicator'></column>
      <column name='EdiDetailLinesProcess'></column>
      <column name='EdiSuccessfullyProcess'></column>
      <column name='TradingPartnerId'></column>
      <column name='TransactionAction'></column>
      <column name='AddressNumber'></column>
      <column name='AlternateAddressKey'></column>
      <column name='TaxId'></column>
      <column name='NameAlpha'></column>
      <column name='DescripCompressed'></column>
      <column name='CostCenter'></column>
      <column name='StandardIndustryCode'></column>
      <column name='LanguagePreference'></column>
      <column name='AddressType1'></column>
      <column name='CreditMessage'></column>
      <column name='PersonCorporationCode'></column>
      <column name='AddressType2'></column>
      <column name='AddressType3'></column>
      <column name='AddressType4'></column>
      <column name='AddressTypeReceivables'></column>
      <column name='AddressType5'></column>
      <column name='AddressTypePayables'></column>
      <column name='AddTypeCode4Purch'></column>
    </table>
  </transaction>
</jdeResponse>
```

```
<column name='MiscCode3'></column>
<column name='AddressTypeEmployee'></column>
<column name='SubledgerInactiveCode'></column>
<column name='DateBeginningEffective'></column>
<column name='AddressNumber1st'></column>
<column name='AddressNumber2nd'></column>
<column name='AddressNumber3rd'></column>
<column name='AddressNumber4th'></column>
<column name='AddressNumber6th'></column>
<column name='AddressNumber5th'></column>
<column name='ReportCodeAddBook001'></column>
<column name='ReportCodeAddBook002'></column>
<column name='ReportCodeAddBook003'></column>
<column name='ReportCodeAddBook004'></column>
<column name='ReportCodeAddBook005'></column>
<column name='ReportCodeAddBook006'></column>
<column name='ReportCodeAddBook007'></column>
<column name='ReportCodeAddBook008'></column>
<column name='ReportCodeAddBook009'></column>
<column name='ReportCodeAddBook010'></column>
<column name='ReportCodeAddBook011'></column>
<column name='ReportCodeAddBook012'></column>
<column name='ReportCodeAddBook013'></column>
<column name='ReportCodeAddBook014'></column>
<column name='ReportCodeAddBook015'></column>
<column name='ReportCodeAddBook016'></column>
<column name='ReportCodeAddBook017'></column>
<column name='ReportCodeAddBook018'></column>
<column name='ReportCodeAddBook019'></column>
<column name='ReportCodeAddBook020'></column>
<column name='CategoryCodeAddressBook2'></column>
<column name='CategoryCodeAddressBk22'></column>
<column name='CategoryCodeAddressBk23'></column>
<column name='CategoryCodeAddressBk24'></column>
<column name='CategoryCodeAddressBk25'></column>
<column name='CategoryCodeAddressBk26'></column>
<column name='CategoryCodeAddressBk27'></column>
<column name='CategoryCodeAddressBk28'></column>
<column name='CategoryCodeAddressBk29'></column>
<column name='CategoryCodeAddressBk30'></column>
```

```
<column name='GIBankAccount'></column>
<column name='TimeScheduledIn'></column>
<column name='DateScheduledIn'></column>
<column name='ActionMessageControl'></column>
<column name='NameRemark'></column>
<column name='CertificateTaxExempt'></column>
<column name='TaxId2'></column>
<column name='Kanjialpha'></column>
<column name='UserReservedCode'></column>
<column name='UserReservedDate'></column>
<column name='UserReservedAmount'></column>
<column name='UserReservedNumber'></column>
<column name='UserReservedReference'></column>
<column name='NameMailing'></column>
<column name='SecondaryMailingName'></column>
<column name='AddressLine1'></column>
<column name='AddressLine2'></column>
<column name='AddressLine3'></column>
<column name='AddressLine4'></column>
<column name='ZipCodePostal'></column>
<column name='City'></column>
<column name='Country'></column>
<column name='State'></column>
<column name='CountyAddress'></column>
<column name='PhoneAreaCode1'></column>
<column name='PhoneNumber'></column>
<column name='PhoneNumberTyp1'></column>
<column name='PhoneAreaCode2'></column>
<column name='PhoneNumber1'></column>
<column name='PhoneNumberTyp2'></column>
<column name='TransactionOriginator'></column>
<column name='UserId'></column>
<column name='ProgramId'></column>
<column name='WorkStationId'></column>
<column name='DateUpdated'></column>
<column name='TimeOfDay'></column>
<column name='TimeLastUpdated'></column>
</table>
</transaction>
</jdeResponse>
```

XML Format Examples (Default Values)

XML Format Examples (Default Values)

The following examples are used for specific formats.

Inbound Sales Order XML Format

This example uses the OneWorld default values. It omits the parameters that an external entity chooses not to fill.

```
<?xml version="1.0" encoding="utf-8" ?>
=<jdeRequest type="callmethod" user="JDE" pwd="JDE" environment="PRD733">
  =<callMethod name="GetLocalComputerId" app="NetComm" runOnError="no">
    =<params>
      <param name="szMachineKey" id="m2" />
    </params>
    <onError abort="yes" />
  </callMethod>
  =<callMethod name="F4211FSBeginDoc" app="NetComm" runOnError="no">
    =<params>
      <param name="mnCMJobNumber" id="j1" />
      <param name="cCMDocAction">A</param>
      <param name="cCMProcessEdits">1</param>
      <param name="szCMComputerID" idref="c2" />
      <param name="cCMUpdateWriteToWF">2</param>
      <param name="szCMProgramID">NetComm</param>
      <param name="szCMVersion">ZJDE0001</param>
      <param name="szOrderType">SO</param>
      <param name="szBusinessUnit">M30</param>
      <param name="mnAddressNumber">4242</param>
      <param name="jdOrderDate">2000/01/21</param>
      <param name="szReference">10261</param>
      <param name="cApplyFreightYN">Y</param>
      <param name="szCurrencyCode">CAD</param>
      <param name="cWKSsourceOfData" />
      <param name="cWKProcMode" />
      <param name="mnWKSsuppressProcess">0</param>
    </params>
    =<onError abort="yes">
```

```

=<callMethod name="F4211ClearWorkFile" app="NetComm" runOnEr-
  ror="yes">
  =<params>
    <param name="mnJobNo" idref="j1" />
    <param name="szComputerID" idref="c2" />
    <param name="mnFromLineNo">0</param>
    <param name="mnThruLineNo">0</param>
    <param name="cClearHeaderWF">2</param>
    <param name="cClearDetailWF">2</param>
    <param name="szProgramID">NetComm</param>
    <param name="szCMVersion">ZJDE0001</param>
  </params>
</callMethod>
</onError>
</callMethod>
=<callMethod name="F4211FSEditLine" app="NetComm" runOnError="yes">
  =<params>
    <param name="mnCMJobNo" idref="j1" />
    <param name="cMLineAction">A</param>
    <param name="cMProcessEdits">1</param>
    <param name="cCMWriteToWFFlag">2</param>
    <param name="szCMComputerID" idref="c2" />
    <param name="szItemNo">1001</param>
    <param name="mnQtyOrdered">1</param>
    <param name="cSalesTaxableYN">N</param>
    <param name="szTransactionUOM">EA</param>
    <param name="szCMProgramID">NetComm</param>
    <param name="szCMVersion">ZJDE0001</param>
    <param name="cWKSourceOfData" />
  </params>
  <onError abort="no" />
</callMethod>
=<callMethod name="F4211FSEditLine" app="NetComm" runOnError="yes">
  =<params>
    <param name="mnCMJobNo" idref="j1" />
    <param name="cMLineAction">A</param>
    <param name="cMProcessEdits">1</param>
    <param name="cCMWriteToWFFlag">2</param>
    <param name="szCMComputerID" idref="c2" />
    <param name="szItemNo">1001</param>
    <param name="mnQtyOrdered">10</param>
    <param name="cSalesTaxableYN">N</param>
    <param name="szTransactionUOM">EA</param>
    <param name="szCMProgramID">NetComm</param>
    <param name="szCMVersion">ZJDE0001</param>
    <param name="cWKSourceOfData" />
  </params>
  <onError abort="no" />

```



```

</callMethod>
=<callMethod name="F4211FSEndDoc" app="NetComm" runOnError="no">
  =<params>
    <param name="mnCMJobNo" idref="j1" />
    <param name="szCMComputerID" idref="c2" />
    <param name="szCMProgramID">NetComm</param>
    <param name="szCMVersion">ZJDE0001</param>
    <param name="cCMUseWorkFiles">2</param>
  </params>
  =<onError abort="no">
    =<callMethod name="F4211ClearWorkFile" app="NetComm" runOnError="yes">
      =<params>
        <param name="mnJobNo" idref="j1" />
        <param name="szComputerID" idref="c2" />
        <param name="mnFromLineNo">0</param>
        <param name="mnThruLineNo">0</param>
        <param name="cClearHeaderWF">2</param>
        <param name="cClearDetailWF">2</param>
        <param name="szProgramID">NetComm</param>
        <param name="szCMVersion">ZJDE0001</param>
      </params>
    </callMethod>
  </onError>
</callMethod>
<returnParams failureDestination="ERROR.Q" successDestination="SUCCESS.Q" runOnError="yes" />
=<onError abort="yes">
  =<callMethod name="F4211ClearWorkFile" app="NetComm" runOnError="yes">
    =<params>
      <param name="mnJobNo" idref="j1" />
      <param name="szComputerID" idref="c2" />
      <param name="mnFromLineNo">0</param>
      <param name="mnThruLineNo">0</param>
      <param name="cClearHeaderWF">2</param>
      <param name="cClearDetailWF">2</param>
      <param name="szProgramID">NetComm</param>
      <param name="szCMVersion">ZJDE0001</param>
    </params>
  </callMethod>
</onError>
</jdeRequest>

```


Glossary

Glossary

AAI. See automatic accounting instruction.

action message. With OneWorld, users can receive messages (system-generated or user-generated) that have shortcuts to OneWorld forms, applications, and appropriate data. For example, if the general ledger post sends an action error message to a user, that user can access the journal entry (or entries) in error directly from the message. This is a central feature of the OneWorld workflow strategy. Action messages can originate either from OneWorld or from a third-party e-mail system.

activator. In the Solution Explorer, a parent task with sequentially-arranged child tasks that are automated with a director.

ActiveX. A computing technology, based on object linking and embedding, that enables Java applet-style functionality for Web browsers as well as other applications. (Java is limited to Web browsers at this time.) The ActiveX equivalent of a Java applet is an ActiveX control. These controls bring computational, communications, and data manipulation power to programs that can “contain” them. For example, certain Web browsers, Microsoft Office programs, and anything developed with Visual Basic or Visual C++.

advance. A change in the status of a project in the Object Management Workbench. When you advance a project, the status change might trigger other actions and conditions such as moving objects from one server to another or preventing check-out of project objects.

alphanumeric character. A combination of letters, numbers, and symbols used to represent data. Contrast with numeric character and special character.

API. See application programming interface.

APPL. See application.

applet. A small application, such as a utility program or a limited-function spreadsheet. It is generally associated with the programming language Java, and in this context refers to

Internet-enabled applications that can be passed from a Web browser residing on a workstation.

application. In the computer industry, the same as an executable file. In OneWorld, an interactive or batch application is a DLL that contains programming for a set of related forms that can be run from a menu to perform a business task such as Accounts Payable and Sales Order Processing. Also known as system.

application developer. A programmer who develops OneWorld applications using the OneWorld toolset.

application programming interface (API). A software function call that can be made from a program to access functionality provided by another program.

application workspace. The area on a workstation display in which all related forms within an application appear.

audit trail. The detailed, verifiable history of a processed transaction. The history consists of the original documents, transaction entries, and posting of records, and usually concludes with a report.

automatic accounting instruction (AAI). A code that refers to an account in the chart of accounts. AAIs define rules for programs that automatically generate journal entries, including interfaces between Accounts Payable, Accounts Receivable, Financial Reporting, General Accounting systems. Each system that interfaces with the General Accounting system has AAIs. For example, AAIs can direct the General Ledger Post program to post a debit to a specific expense account and a credit to a specific accounts payable account.

batch header. The information that identifies and controls a batch of transactions or records.

batch job. A task or group of tasks you submit for processing that the system treats as a single unit during processing, for example, printing reports and purging files. The computer system

performs a batch job with little or no user interaction.

batch processing. A method by which the system selects jobs from the job queue, processes them, and sends output to the outqueue. Contrast with interactive processing.

batch server. A server on which OneWorld batch processing requests (also called UBEs) are run instead of on a client, an application server, or an enterprise server. A batch server typically does not contain a database nor does it run interactive applications.

batch type. A code assigned to a batch job that designates to which J.D. Edwards system the associated transactions pertain, thus controlling which records are selected for processing. For example, the Post General Journal program selects for posting only unposted transaction batches with a batch type of O.

batch-of-one immediate. A transaction method that allows a client application to perform work on a client workstation, then submit the work all at once to a server application for further processing. As a batch process is running on the server, the client application can continue performing other tasks. See also direct connect, store and forward.

BDA. See Business View Design Aid.

binary string (BSTR). A length prefixed string used by OLE automation data manipulation functions. Binary Strings are wide, double-byte (Unicode) strings on 32-bit Windows platforms.

Boolean Logic Operand. In J.D. Edwards reporting programs, the parameter of the Relationship field. The Boolean logic operand instructs the system to compare certain records or parameters. Available options are:

EQ	Equal To.
LT	Less Than.
LE	Less Than or Equal To.
GT	Greater Than.
GE	Greater Than or Equal To.
NE	Not Equal To.
NL	Not Less Than.
NG	Not Greater Than.

browser. A client application that translates information sent by the World Wide Web. A client must use a browser to receive, manipulate, and display World Wide Web

information on the desktop. Also known as a Web browser.

BSFN. See business function.

BSTR. See binary string.

BSVW. See business view.

business function. An encapsulated set of business rules and logic that can normally be reused by multiple applications. Business functions can execute a transaction or a subset of a transaction (check inventory, issue work orders, and so on). Business functions also contain the APIs that allow them to be called from a form, a database trigger, or a non-OneWorld application. Business functions can be combined with other business functions, forms, event rules, and other components to make up an application. Business functions can be created through event rules or third-generation languages, such as C. Examples of business functions include Credit Check and Item Availability.

business function event rule. See named event rule.

business view. Used by OneWorld applications to access data from database tables. A business view is a means for selecting specific columns from one or more tables whose data will be used in an application or report. It does not select specific rows and does not contain any physical data. It is strictly a view through which data can be handled.

Business View Design Aid (BDA). A OneWorld GUI tool for creating, modifying, copying, and printing business views. The tool uses a graphical user interface.

category code. In user defined codes, a temporary title for an undefined category. For example, if you are adding a code that designates different sales regions, you could change category code 4 to Sales Region, and define E (East), W (West), N (North), and S (South) as the valid codes. Sometimes referred to as reporting codes.

central objects. Objects that reside in a central location and consist of two parts: the central objects data source and central C components. The central objects data source contains OneWorld specifications, which are stored in a relational database. Central C components

contain business function source, header, object, library, and DLL files and are usually stored in directories on the deployment server. Together they make up central objects.

check-in location. The directory structure location for the package and its set of replicated objects. This is usually \\deploymentserver\release\path_code\package\packagename. The sub-directories under this path are where the central C components (source, include, object, library, and DLL file) for business functions are stored.

child. See parent/child form.

client/server. A relationship between processes running on separate machines. The server process is a provider of software services. The client is a consumer of those services. In essence, client/server provides a clean separation of function based on the idea of service. A server can service many clients at the same time and regulate their access to shared resources. There is a many-to-one relationship between clients and a server, respectively. Clients always initiate the dialog by requesting a service. Servers passively wait for requests from clients.

CNC. See configurable network computing.

component. In the ActivEra Portal, an encapsulated object that appears inside a workspace. Portal components

configurable client engine. Allows user flexibility at the interface level. Users can easily move columns, set tabs for different data views, and size grids according to their needs. The configurable client engine also enables the incorporation of Web browsers in addition to the Windows 95- and Windows NT-based interfaces.

configurable network computing. An application architecture that allows interactive and batch applications, composed of a single code base, to run across a TCP/IP network of multiple server platforms and SQL databases. The applications consist of reusable business functions and associated data that can be configured across the network dynamically. The overall objective for businesses is to provide a future-proof environment that enables them to change organizational structures, business

processes, and technologies independently of each other.

constants. Parameters or codes that you set and the system uses to standardize information processing by associated programs. Some examples of constants are: validating bills of material online and including fixed labor overhead in costing.

control. Any data entry point allowing the user to interact with an application. For example, check boxes, pull-down lists, hyper-buttons, entry fields, and similar features are controls.

core. The central and foundation systems of J.D. Edwards software, including General Accounting, Accounts Payable, Accounts Receivable, Address Book, Financial Reporting, Financial Modeling and Allocations, and Back Office.

CRP. Conference Room Pilot.

custom gridlines. A grid row that does not come from the database, for example, totals. To display a total in a grid, sum the values and insert a custom gridline to display the total. Use the system function Insert Grid Row Buffer to accomplish this.

data dictionary. The OneWorld method for storing and managing data item definitions and specifications. J.D. Edwards has an active data dictionary, which means it is accessed at runtime.

data mart. Department-level decision support databases. They usually draw their data from an enterprise data warehouse that serves as a source of consolidated and reconciled data from around the organization. Data marts can be either relational or multidimensional databases.

data replication. In a replicated environment, multiple copies of data are maintained on multiple machines. There must be a single source that "owns" the data. This ensures that the latest copy of data can be applied to a primary place and then replicated as appropriate. This is in contrast to a simple copying of data, where the copy is not maintained from a central location, but exists independently of the source.

data source. A specific instance of a database management system running on a computer.

Data source management is accomplished through Object Configuration Manager (OCM) and Object Map (OM).

data structure. A group of data items that can be used for passing information between objects, for example, between two forms, between forms and business functions, or between reports and business functions.

data warehouse. A database used for reconciling and consolidating data from multiple databases before it is distributed to data marts for department-level decision support queries and reports. The data warehouse is generally a large relational database residing on a dedicated server between operational databases and the data marts.

data warehousing. Essentially, data warehousing involves off-loading operational data sources to target databases that will be used exclusively for decision support (reports and queries). There are a range of decision support environments, including duplicated database, enhanced analysis databases, and enterprise data warehouses.

database. A continuously updated collection of all information a system uses and stores. Databases make it possible to create, store, index, and cross-reference information online.

database driver. Software that connects an application to a specific database management system.

database server. A server that stores data. A database server does not have OneWorld logic.

DCE. See distributed computing environment.

DD. See data dictionary.

default. A code, number, or parameter value that is assumed when none is specified.

detail. The specific pieces of information and data that make up a record or transaction. Contrast with summary.

detail area. A control that is found in OneWorld applications and functions similarly to a spreadsheet grid for viewing, adding, or updating many rows of data at one time.

direct connect. A transaction method in which a client application communicates interactively

and directly with a server application. See also batch-of-one immediate, store and forward.

director. An interactive utility that guides a user through the steps of a process to complete a task.

distributed computing environment (DCE). A set of integrated software services that allows software running on multiple computers to perform in a manner that is seamless and transparent to the end-users. DCE provides security, directory, time, remote procedure calls, and files across computers running on a network.

DLL. See dynamic link library.

DS. See data structure.

DSTR. See data structure.

duplicated database. A decision support database that contains a straightforward copy of operational data. The advantages involve improved performance for both operational and reporting environments. See also enhanced analysis database, enterprise data warehouse.

dynamic link library (DLL). A set of program modules that are designed to be invoked from executable files when the executable files are run, without having to be linked to the executable files. They typically contain commonly used functions.

dynamic partitioning. The ability to dynamically distribute logic or data to multiple tiers in a client/server architecture.

embedded event rule. An event rule that is specific to a particular table or application. Examples include form-to-form calls, hiding a field based on a processing option value, and calling a business function. Contrast with business function event rule. See also event rule.

employee work center. This is a central location for sending and receiving all OneWorld messages (system and user generated) regardless of the originating application or user. Each user has a mailbox that contains workflow and other messages, including Active Messages. With respect to workflow, the Message Center is MAPI compliant and supports drag and drop work reassignment, escalation, forward and reply, and workflow monitoring. All messages

from the message center can be viewed through OneWorld messages or Microsoft Exchange.

encapsulation. The ability to confine access to and manipulation of data within an object to the procedures that contribute to the definition of that object.

enhanced analysis database. A database containing a subset of operational data. The data on the enhanced analysis database performs calculations and provides summary data to speed generation of reports and query response times. This solution is appropriate when external data must be added to source data, or when historical data is necessary for trend analysis or regulatory reporting. See also duplicated database, enterprise data warehouse.

enterprise data warehouse. A complex solution that involves data from many areas of the enterprise. This environment requires a large relational database (the data warehouse) that is a central repository of enterprise data, which is clean, reconciled, and consolidated. From this repository, data marts retrieve data to provide department-level decisions. See also duplicated database, enhanced analysis database.

enterprise server. A database server and logic server. See database server. Also referred to as host.

ER. See event rule.

ERP. See enterprise resource planning.

event. An action that occurs when an interactive or batch application is running. Example events are tabbing out of an edit control, clicking a push button, initializing a form, or performing a page break on a report. The GUI operating system uses miniprograms to manage user activities within a form. Additional logic can be attached to these miniprograms and used to give greater functionality to any event within a OneWorld application or report using event rules.

event rule. Used to create complex business logic without the difficult syntax that comes with many programming languages. These logic statements can be attached to applications or database events and are executed when the defined event occurs, such as entering a form, selecting a menu bar option, page breaking on

a report, or selecting a record. An event rule can validate data, send a message to a user, call a business function, as well as many other actions. There are two types of event rules:

- 1 Embedded event rules.
- 2 Named event rules.

executable file. A computer program that can be run from the computer's operating system. Equivalent terms are "application" and "program."

exit. 1) To interrupt or leave a computer program by pressing a specific key or a sequence of keys. 2) An option or function key displayed on a form that allows you to access another form.

facility. 1) A separate entity within a business for which you want to track costs. For example, a facility might be a warehouse location, job, project, work center, or branch/plant. Sometimes referred to as a business unit. 2) In Home Builder and ECS, a facility is a collection of computer language statements or programs that provide a specialized function throughout a system or throughout all integrated systems. For example, DREAM Writer and FASTR are facilities.

FDA. See Form Design Aid.

find/browse. A type of form used to:

- 1 Search, view, and select multiple records in a detail area.
- 2 Delete records.
- 3 Exit to another form.
- 4 Serve as an entry point for most applications.

firewall. A set of technologies that allows an enterprise to test, filter, and route all incoming messages. Firewalls are used to keep an enterprise secure.

fix/inspect. A type of form used to view, add, or modify existing records. A fix/inspect form has no detail area.

form. An element of OneWorld's graphical user interface that contains controls by which a user can interact with an application. Forms allow the user to input, select, and view information. A OneWorld application might contain multiple forms. In Microsoft Windows terminology, a form is known as a dialog box.

Form Design Aid (FDA). The OneWorld GUI development tool for building interactive applications and forms.

form interconnection. Allows one form to access and pass data to another form. Form interconnections can be attached to any event; however, they are normally used when a button is clicked.

form type. The following form types are available in OneWorld:

- 1 Find/browse.
- 2 Fix/inspect.
- 3 Header detail.
- 4 Headerless detail.
- 5 Message.
- 6 Parent/child.
- 7 Search/select.

fourth generation language (4GL). A programming language that focuses on what you need to do and then determines how to do it. Structured Query Language is an example of a 4GL.

graphical user interface (GUI). A computer interface that is graphically based as opposed to being character-based. An example of a character-based interface is that of the AS/400. An example of a GUI is Microsoft Windows. Graphically based interfaces allow pictures and other graphic images to be used in order to give people clues on how to operate the computer.

grid. See detail area.

GUI. See graphical user interface.

header. Information at the beginning of a table or form. This information is used to identify or provide control information for the group of records that follows.

header/detail. A type of form used to add, modify, or delete records from two different tables. The tables usually have a parent/child relationship.

headerless detail. A type of form used to work with multiple records in a detail area. The detail area is capable of receiving input.

hidden selections. Menu selections you cannot see until you enter HS in a menu's Selection field. Although you cannot see these selections, they are available from any menu. They include such items as Display Submitted Jobs (33), Display User Job Queue (42), and

Display User Print Queue (43). The Hidden Selections window displays three categories of selections: user tools, operator tools, and programmer tools.

host. In the centralized computer model, a large timesharing computer system that terminals communicate with and rely on for processing. It contrasts with client/server in that those users work at computers that perform much of their own processing and access servers that provide services such as file management, security, and printer management.

HTML. See hypertext markup language.

hypertext markup language. A markup language used to specify the logical structure of a document rather than the physical layout. Specifying logical structure makes any HTML document platform independent. You can view an HTML document on any desktop capable of supporting a browser. HTML can include active links to other HTML documents anywhere on the Internet or on intranet sites.

index. Represents both an ordering of values and a uniqueness of values that provide efficient access to data in rows of a table. An index is made up of one or more columns in the table.

inheritance. The ability of a class to receive all or parts of the data and procedure definitions from a parent class. Inheritance enhances development through the reuse of classes and their related code.

install system code. See system code.

integrated toolset. Unique to OneWorld is an industrial-strength toolset embedded in the already comprehensive business applications. This toolset is the same toolset used by J.D. Edwards to build OneWorld interactive and batch applications. Much more than a development environment, however, the OneWorld integrated toolset handles reporting and other batch processes, change management, and basic data warehousing facilities.

interactive processing. Processing actions that occur in response to commands you enter directly into the system. During interactive processing, you are in direct communication with the system, and it might prompt you for additional information while processing your

request. See also online. Contrast with batch processing.

interface. A link between two or more computer systems that allows these systems to send information to and receive information from one another.

Internet. The worldwide constellation of servers, applications, and information available to a desktop client through a phone line or other type of remote access.

interoperability. The ability of different computer systems, networks, operating systems, and applications to work together and share information.

intranet. A small version of the Internet usually confined to one company or organization. An intranet uses the functionality of the Internet and places it at the disposal of a single enterprise.

IP. A connection-less communication protocol that by itself provides a datagram service. Datagrams are self-contained packets of information that are forwarded by routers based on their address and the routing table information contained in the routers. Every node on a TCP/IP network requires an address that identifies both a network and a local host or node on the network. In most cases the network administrator sets up these addresses when installing new workstations. In some cases, however, it is possible for a workstation, when booting up, to query a server for a dynamically assigned address.

IServer Service. Developed by J.D. Edwards, this internet server service resides on the web server, and is used to speed up delivery of the Java class files from the database to the client.

ISO 9000. A series of standards established by the International Organization for Standardization, designed as a measure of product and service quality.

J.D. Edwards Database. See JDEBASE Database Middleware.

Java. An Internet executable language that, like C, is designed to be highly portable across platforms. This programming language was developed by Sun Microsystems. Applets, or Java applications, can be accessed from a web browser and executed at the client, provided

that the operating system or browser is Java-enabled. (Java is often described as a scaled-down C++). Java applications are platform independent.

Java Database Connectivity (JDBC). The standard way to access Java databases, set by Sun Microsystems. This standard allows you to use any JDBC driver database.

JavaScript. A scripting language related to Java. Unlike Java, however, JavaScript is not an object-oriented language and it is not compiled.

jde.ini. J.D. Edwards file (or member for AS/400) that provides the runtime settings required for OneWorld initialization. Specific versions of the file/member must reside on every machine running OneWorld. This includes workstations and servers.

JDEBASE Database Middleware. J.D. Edwards proprietary database middleware package that provides two primary benefits:

1. Platform-independent APIs for multidatabase access. These APIs are used in two ways:
 - a. By the interactive and batch engines to dynamically generate platform-specific SQL, depending on the datasource request.
 - b. As open APIs for advanced C business function writing. These APIs are then used by the engines to dynamically generate platform-specific SQL.
2. Client-to-server and server-to-server database access. To accomplish this OneWorld is integrated with a variety of third-party database drivers, such as Client Access 400 and open database connectivity (ODBC).

JDECallObject. An application programming interface used by business functions to invoke other business functions.

JDENET. J.D. Edwards proprietary middleware software. JDENET is a messaging software package.

JDENET communications middleware. J.D. Edwards proprietary communications middleware package for OneWorld. It is a peer-to-peer, message-based, socket based, multiprocess communications middleware solution. It handles client-to-server and

server-to-server communications for all OneWorld supported platforms.

job queue. A group of jobs waiting to be batch processed. See also batch processing.

just in time installation (JITI). OneWorld's method of dynamically replicating objects from the central object location to a workstation.

just in time replication (JITR). OneWorld's method of replicating data to individual workstations. OneWorld replicates new records (inserts) only at the time the user needs the data. Changes, deletes, and updates must be replicated using Pull Replication.

KEY. A column or combination of columns that identify one or more records in a database table.

leading zeros. A series of zeros that certain facilities in J.D. Edwards systems place in front of a value you enter. This normally occurs when you enter a value that is smaller than the specified length of the field. For example, if you enter 4567 in a field that accommodates eight numbers, the facility places four zeros in front of the four numbers you enter. The result appears as: 00004567.

level of detail. 1) The degree of difficulty of a menu in J.D. Edwards software. The levels of detail for menus are as follows:

- A Major Product Directories.
- B Product Groups.
- 1 Basic Operations.
- 2 Intermediate Operations.
- 3 Advanced Operations.
- 4 Computer Operations.
- 5 Programmers.
- 6 Advanced Programmers Also known as menu levels.

2) The degree to which account information in the General Accounting system is summarized. The highest level of detail is 1 (least detailed) and the lowest level of detail is 9 (most detailed).

MAPI. See Messaging Application Programming Interface.

master table. A database table used to store data and information that is permanent and necessary to the system's operation. Master tables might contain data such as paid tax

amounts, supplier names, addresses, employee information, and job information.

menu. A menu that displays numbered selections. Each of these selections represents a program or another menu. To access a selection from a menu, type the selection number and then press Enter.

menu levels. See level of detail.

menu masking. A security feature of J.D. Edwards systems that lets you prevent individual users from accessing specified menus or menu selections. The system does not display the menus or menu selections to unauthorized users.

Messaging Application Programming Interface (MAPI). An architecture that defines the components of a messaging system and how they behave. It also defines the interface between the messaging system and the components.

middleware. A general term that covers all the distributed software needed to support interactions between clients and servers. Think of it as the software that's in the middle of the client/server system or the "glue" that lets the client obtain a service from a server.

modal. A restrictive or limiting interaction created by a given condition of operation. Modal often describes a secondary window that restricts a user's interaction with other windows. A secondary window can be modal with respect to its primary window or to the entire system. A modal dialog box must be closed by the user before the application continues.

mode. In reference to forms in OneWorld, mode has two meanings:

- An operational qualifier that governs how the form interacts with tables and business views. OneWorld form modes are: add, copy, and update.
- An arbitrary setting that aids in organizing form generation for different environments. For example, you might set forms generated for a Windows environment to mode 1 and forms generated for a Web environment to mode 2.

modeless. Not restricting or limiting interaction. Modeless often describes a secondary window that does not restrict a user's interaction with

other windows. A modeless dialog box stays on the screen and is available for use at any time but also permits other user activities.

multitier architecture. A client/server architecture that allows multiple levels of processing. A tier defines the number of computers that can be used to complete some defined task.

named event rule. Encapsulated, reusable business logic created using through event rules rather than C programming. Contrast with embedded event rule. See also event rule.

NER. See named event rule.

network computer. As opposed to the personal computer, the network computer offers (in theory) lower cost of purchase and ownership and less complexity. Basically, it is a scaled-down PC (very little memory or disk space) that can be used to access network-based applications (Java applets, ActiveX controls) via a network browser.

network computing. Often referred to as the next phase of computing after client/server. While its exact definition remains obscure, it generally encompasses issues such as transparent access to computing resources, browser-style front-ends, platform independence, and other similar concepts.

next numbers. A feature you use to control the automatic numbering of such items as new G/L accounts, vouchers, and addresses. It lets you specify a numbering system and provides a method to increment numbers to reduce transposition and typing errors.

non-object librarian object. An object that is not managed by the object librarian.

numeric character. Digits 0 through 9 that are used to represent data. Contrast with alphanumeric characters.

object. A self-sufficient entity that contains data as well as the structures and functions used to manipulate the data. For OneWorld purposes, an object is a reusable entity that is based on software specifications created by the OneWorld toolset. See also object librarian.

object configuration manager (OCM). OneWorld's Object Request Broker and the control center for the runtime environment. It keeps track of the runtime locations for

business functions, data, and batch applications. When one of these objects is called, the Object Configuration Manager directs access to it using defaults and overrides for a given environment and user.

object embedding. When an object is embedded in another document, an association is maintained between the object and the application that created it; however, any changes made to the object are also only kept in the compound document. See also object linking.

object librarian. A repository of all versions, applications, and business functions reusable in building applications. You access these objects with the Object Management Workbench.

object librarian object. An object managed by the object librarian.

object linking. When an object is linked to another document, a reference is created with the file the object is stored in, as well as with the application that created it. When the object is modified, either from the compound document or directly through the file it is saved in, the change is reflected in that application as well as anywhere it has been linked. See also object embedding.

object linking and embedding (OLE). A way to integrate objects from diverse applications, such as graphics, charts, spreadsheets, text, or an audio clip from a sound program. See also object embedding, object linking.

object management workbench (OMW). An application that provides check-out and check-in capabilities for developers, and aids in the creation, modification, and use of OneWorld Objects. The OMW supports multiple environments (such as production and development).

object-based technology (OBT). A technology that supports some of the main principles of object-oriented technology: classes, polymorphism, inheritance, or encapsulation.

object-oriented technology (OOT). Brings software development past procedural programming into a world of reusable programming that simplifies development of applications. Object orientation is based on the following principles: classes, polymorphism, inheritance, and encapsulation.

OCM. See object configuration manager.

ODBC. See open database connectivity.

OLE. See object linking and embedding.

OMW. Object Management Workbench.

OneWorld. A combined suite of comprehensive, mission-critical business applications and an embedded toolset for configuring those applications to unique business and technology requirements. OneWorld is built on the Configurable Network Computing technology- J.D. Edwards' own application architecture, which extends client/server functionality to new levels of configurability, adaptability, and stability.

OneWorld application. Interactive or batch processes that execute the business functionality of OneWorld. They consist of reusable business functions and associated data that are platform independent and can be dynamically configured across a TCP/IP network.

OneWorld object. A reusable piece of code that is used to build applications. Object types include tables, forms, business functions, data dictionary items, batch processes, business views, event rules, versions, data structures, and media objects. See also object.

OneWorld process. Allows OneWorld clients and servers to handle processing requests and execute transactions. A client runs one process, and servers can have multiple instances. OneWorld processes can also be dedicated to specific tasks (for example, workflow messages and data replication) to ensure that critical processes don't have to wait if the server is particularly busy.

OneWorld Web development computer. A standard OneWorld Windows developer computer with the additional components installed:

- JFC (0.5.1).
- Generator Package with Generator.Java and JDECOM.dll.
- R2 with interpretive and application controls/form.

online. Computer functions over which the system has continuous control. Users are online with the system when working with J.D. Edwards system provided forms.

open database connectivity (ODBC). Defines a standard interface for different technologies to process data between applications and different data sources. The ODBC interface is made up of a set of function calls, methods of connectivity, and representation of data types that define access to data sources.

open systems interconnection (OSI). The OSI model was developed by the International Standards Organization (ISO) in the early 1980s. It defines protocols and standards for the interconnection of computers and network equipment.

operand. See Boolean Logic Operand.

output. Information that the computer transfers from internal storage to an external device, such as a printer or a computer form.

output queue. See print queue.

package. OneWorld objects are installed to workstations in packages from the deployment server. A package can be compared to a bill of material or kit that indicates the necessary objects for that workstation and where on the deployment server the install program can find them. It is a point-in-time "snap shot" of the central objects on the deployment server.

package location. The directory structure location for the package and its set of replicated objects. This is usually \\deployment server\release\path_code\package\ package name. The sub-directories under this path are where the replicated objects for the package will be placed. This is also referred to as where the package is built or stored.

parameter. A number, code, or character string you specify in association with a command or program. The computer uses parameters as additional input or to control the actions of the command or program.

parent/child form. A type of form that presents parent/child relationships in an application on one form. The left portion of the form presents a tree view that displays a visual representation of a parent/child relationship. The right portion of the form displays a detail area in browse mode. The detail area displays the records for the child item in the tree. The parent/child form supports drag and drop functionality.

partitioning. A technique for distributing data to local and remote sites to place data closer to the users who access. Portions of data can be copied to different database management systems.

path code. A pointer to a specific set of objects. A path code is used to locate:

1. Central Objects.
2. Replicated Objects.

platform independence. A benefit of open systems and Configurable Network Computing. Applications that are composed of a single code base can be run across a TCP/IP network consisting of various server platforms and SQL databases.

polymorphism. A principle of object-oriented technology in which a single mnemonic name can be used to perform similar operations on software objects of different types.

portability. Allows the same application to run on different operating systems and hardware platforms.

portal. A configurable Web object that provides information and links to the Web. Portals can be used as home pages and are typically used in conjunction with a Web browser.

primary key. A column or combination of columns that uniquely identifies each row in a table.

print queue. A list of tables, such as reports, that you have submitted to be written to an output device, such as a printer. The computer spools the tables until it writes them. After the computer writes the table, the system removes the table identifier from the list.

processing option. A feature of the J.D. Edwards reporting system that allows you to supply parameters to direct the functions of a program. For example, processing options allow you to specify defaults for certain form displays, control the format in which information prints on reports, change how a form displays information, and enter beginning dates.

program temporary fix (PTF). A representation of changes to J.D. Edwards software that your organization receives on magnetic tapes or diskettes.

project. An Object Management Workbench object used to organize objects in development.

published table. Also called a “Master” table, this is the central copy to be replicated to other machines. Resides on the “Publisher” machine. the Data Replication Publisher Table (F98DRPUB) identifies all of the Published Tables and their associated Publishers in the enterprise.

publisher. The server that is responsible for the Published Table. The Data Replication Publisher Table (F98DRPUB) identifies all of the Published Tables and their associated Publishers in the enterprise.

pull replication. One of the OneWorld methods for replicating data to individual workstations. Such machines are set up as Pull Subscribers using OneWorld’s data replication tools. The only time Pull Subscribers are notified of changes, updates, and deletions is when they request such information. The request is in the form of a message that is sent, usually at startup, from the Pull Subscriber to the server machine that stores the Data Replication Pending Change Notification table (F98DRPCN).

purge. The process of removing records or data from a system table.

QBE. See query by example.

query by example (QBE). Located at the top of a detail area, it is used to search for data to be displayed in the detail area.

redundancy. Storing exact copies of data in multiple databases.

regenerable. Source code for OneWorld business functions can be regenerated from specifications (business function names). Regeneration occurs whenever an application is recompiled, either for a new platform or when new functionality is added.

relationship. Links tables together and facilitates joining business views for use in an application or report. Relationships are created based on indexes.

release/release update. A “release” contains major new functionality, and a “release update” contains an accumulation of fixes and performance enhancements, but no new functionality.

replicated object. A copy or replicated set of the central objects must reside on each client

and server that run OneWorld. The path code indicates the directory the directory where these objects are located.

run. To cause the computer system to perform a routine, process a batch of transactions, or carry out computer program instructions.

SAR. See software action request.

scalability. Allows software, architecture, network, or hardware growth that will support software as it grows in size or resource requirements. The ability to reach higher levels of performance by adding microprocessors.

search/select. A type of form used to search for a value and return it to the calling field.

selection. Found on J.D. Edwards menus, selections represent functions that you can access from a menu. To make a selection, type the associated number in the Selection field and press Enter.

server. Provides the essential functions for furnishings services to network users (or clients) and provides management functions for network administrators. Some of these functions are storage of user programs and data and management functions for the file systems. It may not be possible for one server to support all users with the required services. Some examples of dedicated servers that handle specific tasks are backup and archive servers, application and database servers.

servlet. Servlets provide a Java-based solution used to address the problems currently associated with doing server-side programming, including inextensible scripting solutions. Servlets are objects that conform to a specific interface that can be plugged into a Java-based server. Servlets are to the server-side what applets are to the client-side.

software. The operating system and application programs that tell the computer how and what tasks to perform.

software action request (SAR). An entry in the AS/400 database used for requesting modifications to J.D. Edwards software.

special character. A symbol used to represent data. Some examples are *, &, #, and /. Contrast with alphanumeric character and numeric character.

specifications. A complete description of a OneWorld object. Each object has its own specification, or name, which is used to build applications.

Specs. See specifications.

spool. The function by which the system stores generated output to await printing and processing.

spooled table. A holding file for output data waiting to be printed or input data waiting to be processed.

SQL. See structured query language.

static text. Short, descriptive text that appears next to a control variable or field. When the variable or field is enabled, the static text is black; when the variable or field is disabled, the static text is gray.

store and forward. A transaction method that allows a client application to perform work and, at a later time, complete that work by connecting to a server application. This often involves uploading data residing on a client to a server.

structured query language (SQL). A fourth generation language used as an industry standard for relational database access. It can be used to create databases and to retrieve, add, modify, or delete data from databases. SQL is not a complete programming language because it does not contain control flow logic.

subfile. See detail.

submit. See run.

subscriber. The server that is responsible for the replicated copy of a Published Table. Such servers are identified in the Subscriber Table.

subscriber table. The Subscriber Table (F98DRSUB), which is stored on the Publisher Server with the Data Replication Publisher Table (F98DRPUB) identifies all of the Subscriber machines for each Published Table.

subsystem job. Within OneWorld, subsystem jobs are batch processes that continually run independently of, but asynchronously with, OneWorld applications.

summary. The presentation of data or information in a cumulative or totaled manner in which most of the details have been

removed. Many of the J.D. Edwards systems offer forms and reports that are summaries of the information stored in certain tables. Contrast with detail.

system. See application.

System Code. System codes are a numerical representation of J.D. Edwards and customer systems. For example, 01 is the system code for Address Book. System codes 55 through 59 are reserved for customer development by customers. Use system codes to categorize within OneWorld. For example, when establishing user defined codes (UDCs), you must include the system code the best categorizes it. When naming objects such as applications, tables, and menus, the second and third characters in the object's name is the system code for that object. For example, G04 is the main menu for Accounts Payable, and 04 is its system code.

system function. A program module, provided by OneWorld, available to applications and reports for further processing.

table. A two-dimensional entity made up of rows and columns. All physical data in a database are stored in tables. A row in a table contains a record of related information. An example would be a record in an Employee table containing the Name, Address, Phone Number, Age, and Salary of an employee. Name is an example of a column in the employee table.

table design aid (TDA). A OneWorld GUI tool for creating, modifying, copying, and printing database tables.

table event rules. Use table event rules to attach database triggers (or programs) that automatically run whenever an action occurs against the table. An action against a table is referred to as an event. When you create a OneWorld database trigger, you must first determine which event will activate the trigger. Then, use Event Rules Design to create the trigger. Although OneWorld allows event rules to be attached to application events, this functionality is application specific. Table event rules provide embedded logic at the table level.

TAM. Table Access Management.

TBLE. See table.

TC. Table conversion.

TCP/IP. Transmission Control Protocol/Internet Protocol. The original TCP protocol was developed as a way to interconnect networks using many different types of transmission methods. TCP provides a way to establish a connection between end systems for the reliable delivery of messages and data.

TCP/IP services port. Used by a particular server application to provide whatever service the server is designed to provide. The port number must be readily known so that an application programmer can request it by name.

TDA. See table design aid.

TER. See table event rules.

Terminal Identification. The workstation ID number. Terminal number of a specific terminal or IBM user ID of a particular person for whom this is a valid profile. Header Field: Use the Skip to Terminal/User ID field in the upper portion of the form as an inquiry field in which you can enter the number of a terminal or the IBM user ID of a specific person whose profile you want the system to display at the top of the list. When you first access this form, the system automatically enters the user ID of the person signed on to the system. Detail Field: The Terminal/User ID field in the lower portion of the form contains the user ID of the person whose profile appears on the same line. A code identifying the user or terminal for which you accessed this window.

third generation language (3GL). A programming language that requires detailed information about how to complete a task. Examples of 3GLs are COBOL, C, Pascal and FORTRAN.

token. A referent to an object used to determine ownership of that object and to prevent non-owners from checking the object out in Object Management Workbench. An object holds its own token until the object is checked out, at which time the object passes its token to the project in which the object is placed.

trigger. Allow you to attach default processing to a data item in the data dictionary. When that data item is used on an application or report, the trigger is invoked by an event associated with the data item. OneWorld also has three

visual assist triggers: calculator, calendar and search form.

UBE. Universal batch engine.

UDC Edit Control. Use a User-Defined Code (UDC) Edit Control for a field that accepts only specific values defined in a UDC table.

Associate a UDC edit control with a database item or dictionary item. The visual assist Flashlight automatically appears adjacent to the UDC edit control field. When you click on the visual assist Flashlight, the attached search and select form displays valid values for the field.

To create a UDC Edit Control, you must:

- Associate the data item with a specific UDC table in the Data Dictionary.
- Create a search and select form for displaying valid values from the UDC table.

uniform resource identifier (URI). A character string that references an internet object by name or location. A URL is a type of URI.

uniform resource locator (URL). Names the address (location) of a document on the Internet or an intranet. A URL includes the document's protocol and server name. The path to the document might be included as well. The following is an example of a URL:
<http://www.jdedwards.com>. This is J.D. Edwards Internet address.

URI. See uniform resource identifier.

URL. See uniform resource locator.

user defined code (type). The identifier for a table of codes with a meaning you define for the system, such as ST for the Search Type codes table in Address Book. J.D. Edwards systems provide a number of these tables and allow you to create and define tables of your own. User defined codes were formerly known as descriptive titles.

user defined codes (UDC). Codes within software that users can define, relate to code descriptions, and assign valid values. Sometimes user defined codes are referred to as a generic code table. Examples of such codes are unit-of-measure codes, state names, and employee type codes.

UTB. Universal Table Browser.

valid codes. The allowed codes, amounts, or types of data that you can enter in a field. The system verifies the information you enter against the list of valid codes.

visual assist. Forms that can be invoked from a control to assist the user in determining what data belongs in the control.

vocabulary overrides. A feature you can use to override field, row, or column title text on forms and reports.

wchar_t. Internal type of a wide character. Used for writing portable programs for international markets.

web client. Any workstation that contains an internet browser. The web client communicates with the web server for OneWorld data.

web server. Any workstation that contains the IServer service, SQL server, Java menus and applications, and Internet middleware. The web server receives data from the web client, and passes the request to the enterprise server. When the enterprise server processes the information, it sends it back to the web server, and the web server sends it back to the web client.

WF. See workflow.

window. See form.

workflow. According to the Workflow Management Coalition, workflow means "the automation of a business process, in whole or part, during which documents, information, or tasks are passed from one participant to another for action, according to a set of procedural rules."

workgroup server. A remote database server usually containing subsets of data replicated from a master database server. This server does not performance an application or batch processing. It may or may not have OneWorld running (in order to replicate data).

workspace. In the ActivEra Portal, the main section of the Portal. A user might have access to several workspaces, each one configured differently and containing its own components.

worldwide web. A part of the Internet that can transmit text, graphics, audio, and video. The

World Wide Web allows clients to launch local or remote applications.

z file. For store and forward (network disconnected) user, OneWorld store and forward applications perform edits on static data and other critical information that must be valid to process an order. After the initial edits are complete, OneWorld stores the transactions in work tables on the workstation. These work table are called Z files. When a network connection is established, Z files are uploaded to the enterprise server and the transactions are edited again by a master business function. The master business function will then update the records in your transaction files.

Index

Index

A

- API, completion confirmation, 10–27
- APIs, 1–2
- Asynchronous Transactions
 - from OneWorld, 9–17
 - Into OneWorld, 9–7
- Asynchronous XML Transactions
 - Inbound, 8–7
 - Outbound, 8–7
- Audit Trail report, sample, 11–20

B

- Batch Transactions
 - into OneWorld, 9–11
 - outbound, 9–21
- Business function
 - creating, documentation, C–3, C–4
 - documentation, C–1, C–3
 - generating, C–9
 - template, C–5
 - viewing, C–14
 - generating, documentation, C–9
 - viewing, documentation, C–14
- Business function - values to pass, documentation, viewing, C–15
- Business function document viewer, documentation, viewing, C–16
- Business Function documentation, data selection, C–13
- Business Function error handling, 8–12
 - error text, 8–12
 - ID/IDREF, 8–14
 - multiple requests per document, 8–13
 - on error handling, 8–13
 - return null values, 8–15
- Business function search, documentation, viewing, C–14
- Business function wrappers, generating, 2–2
- Business Functions, calling externally, 10–17

C

- CallObject, 8–9
- CheckVer, running, 3–33
- CheckVer COM, 3–33
- CheckVer CORBA, 4–22
- CheckVer JAVA, 4–21
- Choosing a connector, 2–1
- COM
 - and OneWorld, 3–3
 - connector, 3–6
 - flow in OneWorld, 3–4
 - objects, 3–3
 - OneWorld server, 3–6
 - overview, 3–1, 8–1
 - reliability, 3–7
 - server deployment, 3–9
 - tracing and logging, 3–35
 - understanding, 3–3
 - using BHVRCOM, 3–19
 - version checker, 3–33
- COM CheckVer, 3–33
- Com Server, installing on non-OneWorld machine, 3–25
- Completion Confirmation, API, 10–27
- Connecting, to and from OneWorld, 10–3
- Connectors, 2–1
 - choosing, 2–1
- Copying data into the EDI Outbound Interface tables, 11–7
- CORBA
 - and OneWorld, 5–5
 - overview, 5–1
 - setting up an environment, 5–19
 - understanding, 5–3
- CORBA Server, building and deploying, 5–19
- Corba versioning, 4–19
- Creating
 - business function documentation, C–3, C–4
 - data structure documentation, C–6
 - parameter documentation, C–8

- Creating an XML Template, 8–21
- Creating Transactions
 - From OneWorld, A–17
 - into and from OneWorld, common standards, A–7
 - Into OneWorld, A–13
 - task summary, A–21

D

- Data, formatting, 13–3
- Data Interface, inbound edit/update program, 11–3
- Data structure
 - creating, documentation, C–6
 - documentation, template, C–8
- DCOM
 - client environment, set up, 3–31
 - server, 3–27
 - server environment, set up, 3–27
- DCOM server, security, set up, 3–27
- Defining OneWorld subsystem jobs, 11–11
- Disconnect, from OneWorld, 10–7
- Document processing, inbound, updating applications, 11–3
- Documentation
 - business function, C–1, C–3
 - creating, C–3, C–4
 - template, C–5
 - business function - values to pass, viewing, C–15
 - business function document viewer, viewing, C–16
 - business function search, viewing, C–14
 - data structure
 - creating, C–6
 - template, C–8
 - generating, business function, C–9
 - parameter, creating, C–8
 - viewing, business function, C–14

E

- EDI, overview, 1–9
- Enabling outbound Z table processes, XML, 8–16

- Example
 - adding records to interface table, 10–11
 - calling business functions externally, 10–24
 - connect to OneWorld, 10–5
 - disconnect from OneWorld, 10–8
 - retrieving records from interface tables, 10–16
 - vendor function, 10–28

F

- F4101Z1 Revisions form, 11–22
- Formatting, data, 13–3
- Forms
 - employee work center, 11–20
 - F4101Z1 Revisions, 11–22
 - Work with Batch Versions, 11–4, 11–8, 11–25

G

- GenCOM
 - flow, 3–8
 - OneWorld client environment
 - include directories, 3–21
 - lib directories, 3–22
 - MSDev directories, 3–22
 - paths, 3–22
 - ProgID, 3–15
 - setting up a OneWorld client environment, 3–21
 - using, 3–11
- GenCom
 - description, 5–9
 - options, 3–12
 - output, 3–15
 - running, 3–11
 - syntac, 3–11
- GenCORBA
 - classpath, 5–17
 - options, 5–10
 - path, 5–17
 - running, 5–9
 - setting up a OneWorld client environment, 5–17
 - syntax, 5–9

- understanding, 5–6, 5–9
- GenCorba, output, 5–12
- GenCORBA CheckVer, running, 4–22
- Generating business function wrappers, 2–2
- GenJAVA
 - setting up OneWorld client environment, 4–15
 - understanding, 4–6
- GenJava
 - description, 4–7
 - options, 4–8
 - syntax, 4–7
 - using output, 4–10
- GenJAVA CheckVer, running, 4–21

I

- ID/IDREF, 8–14
- ijDEScript, 6–1
- ijDEScript commands, 6–2
 - Build, 6–2
 - Call, 6–2
 - Define, 6–2
 - Define!, 6–3
 - Exit, 6–3
 - Help, 6–4
 - Import, 6–4
 - Importlib, 6–5
 - Interface, 6–5
 - Library, 6–6
 - Login, 6–6
 - Logout, 6–7
 - Opt, 6–7
 - Rename, 6–7
 - Say, 6–8
 - Sub, 6–8
 - System, 6–9
- Inbound and Outbound Synchronous XML CallObject, 8–6
- Inbound Asynchronous XML Transactions, 8–7
- Inbound edit/update program, 11–3
- Inbound Processor UBE, 11–3
- Inbound Transaction
 - example, 10–31
 - subsystem data structure, 10–29
 - transaction confirmation, 10–30
- Inbound transaction, confirmation, 12–5

- Installing JAVA components on a non-OneWorld machine, 4–17
- Interface Tables
 - adding records, 10–9
 - retrieving records, 10–13
- Interoperability
 - API models, 1–11
 - benefits, 1–1
 - overview, 1–1
 - types, 1–6
- InteropOutBoundConfirmationFunc, 10–27

J

- JAVA
 - and OneWorld, 4–3
 - component usage, 4–6
 - installing components on a non-OneWorld machine, 4–17
 - understanding, 4–3
 - version checker, 4–21
- Java, overview, 4–1
- JAVA CheckVer, 4–21
- Java versioning, 4–19
- jde.ini file, setting for XML, 8–23
- JDEBase database middleware, 1–4
- jdeinterop.ini, 7–1
- jdeinterop.INI files
 - [INTEROP SETTINGS], Understanding jdeinterop.ini Settings, 7–3, 7–4
 - [JDENET SETTINGS], Understanding jdeinterop.ini Settings, 7–1
 - [LOGS], Understanding jdeinterop.ini Settings, 7–2
 - [SERVER SETTINGS], Understanding jdeinterop.ini Settings, 7–1
- JDENet communication middleware, 1–3
- Jobs, subsystem, defining, 11–11

M

- Middleware, 1–3
 - JDEBase database, 1–4
 - JDENet communication, 1–3
- Models
 - Asynchronous From OneWorld, 9–17
 - Asynchronous Into OneWorld, 9–7

- Batch from OneWorld, 9–21
- Batch Into OneWorld, 9–11
- interoperability, 9–1
- Synchronous From OneWorld, 9–15
- Synchronous Into OneWorld, 9–5

Modes, static and dynamic, 4–19

O

ODA

- error messages, D–15
- working with, D–11

ODA Data Source, adding, D–5

OneWorld subsystem jobs, defining for OneWorld, 11–11

Open Data Access, description, D–1

Outbound Asynchronous XML Transactions, 8–7

Outbound Batch, vendor specific, 12–3

Outbound notification, XML, 8–16

Outbound Transactions, subscribing to, 11–13

Outbound transactions, batch, 9–21

P

Parameter, creating, documentation, C–8

Processing, modes, 9–1

R

Running CheckVer, 3–33

Running GenJava, 4–7

S

Scheduler, using, 13–7

Setting an environment for CORBA, 5–19

Setting the jde.ini file for XML, 8–23

Setting up a OneWorld client environment for GenCOM, 3–21

Setting up a OneWorld client environment for GenCORBA, 5–17

Setting up a OneWorld client environment for GenJAVA, 4–15

Setting up a OneWorld DCOM server, 3–27

Subsystem Data Queues, placing entries in, 10–29

Subsystem Jobs, running, 11–11

Subsystem jobs, defining for OneWorld, 11–11

Synchronous Transactions

- from OneWorld, 9–15
- Into OneWorld, 9–5

Synchronous XML CallObject, Inbound and Outbound, 8–6

T

Template

- business function documentation, C–5
- data structure documentation, C–8

ThinNet, 8–4

Transactions from OneWorld, Asynchronous, 9–17

Transactions Into OneWorld

- asynchronous, 9–7
- batch, 9–11
- Synchronous, 9–5
- synchronous, 9–15

U

UBEs, running from menu, 11–7

Understanding COM, 3–3

Understanding CORBA, 5–3

Understanding Java, 4–3

Understanding JAVA ad CORBA versioning, 4–19

Understanding the CORBA generator, 5–9

Understanding XML, 8–3

Updating applications with EDI data, 11–3

Using BHVRCOM via COM, 3–19

Using COM Tracing and Logging, 3–35

Using GenJava output, 4–10

Using the COM wrapper version checker, 3–33

Using the JAVA wrapper version checker, 4–21

V

Versioning, JAVA and CORBA, 4-19

Viewing, documentation

- business function - values to pass, C-15

- business function document viewer, C-16

- business function search, C-14

W

Working with XML CallObject, 8-9

X

XML

- and OneWorld, 8-3

- CallObject templates, 8-21

- enabling outbound Z table processes,
8-16

- format examples, E-1

- outbound notification, 8-16

- setting the jde.ini file, 8-23

- understanding, 8-3

- Z table inquiry API, 8-18

XML CallObject, 8-9

- call object, 8-10

- error handling, 8-12

- establish session, 8-9

- expire session, 8-9

- explicit transaction, 8-10

- implicit transaction, 8-10

- prepare/commit/rollback, 8-11

- terminate session, 8-12

XML CallObject Templates, 8-21

XML format examples, E-1

- all parameters, E-3

- default values, E-19

XML overview, 8-1

XML template, creating, 8-21

XML Z table inquiry API, 8-18

