JD Edwards EnterpriseOne Tools

Interoperability Guide

Release 8.98 Update 4

E14711-02

March 2011

ORACLE®

JD Edwards EnterpriseOne Tools Interoperability Guide, Release 8.98 Update 4

E14711-02

Contents

8 Understanding XML CallObject

9 Understanding XML Transaction

10 Understanding XML List

11 Processing Z Transactions

12 Using Flat Files

13 Understanding Messaging Queue Adapters

14 Using Guaranteed Events

20 Using the Java Database Connectivity Driver

21 Using Oracle Orchestration Systems

22 Setting Up Orchestration Cross-References

A Classic Events

B Using Classic Real-Time Events

C Using Classic XAPI Events

D Using Classic Z Events

E Events Self-Diagnostic Utility Tool

F Interoperability Interface Table Information

G XML Format Examples (All Parameters)

H Minimum Required Values Sample Code

I XML Format Examples (Events)

Glossary

Index

Preface

Welcome to the JD Edwards EnterpriseOne Tools Interoperability Guide.

Audience

This guide is intended for system administrators and technical consultants who are responsible for interoperability.

This guide assumes you have a working knowledge of the following:

• The principles and customary practices of your business area.

• Computer desktop application usage and terminology.

Interoperability Companion Documentation

Additional, essential information describing the setup and design of Oracle's JD Edwards EnterpriseOne Tools Interoperability resides in companion documentation. The companion documentation consists of topics that apply to Interoperability models as well as other JD Edwards EnterpriseOne Tools. Depending on which interoperability model you use, you should be familiar with the information in the companion guide.

- JD Edwards EnterpriseOne Business Services Development Guide
- JD Edwards EnterpriseOne Business Services Development Methodology Guide
- JD Edwards EnterpriseOne Business Services Server Reference Guide
- Web Services Gateway documentation
- JD Edwards EnterpriseOne Tools Connectors Guide
- JD Edwards EnterpriseOne Tools Table Conversion
- JD Edwards EnterpriseOne Tools Output Stream Access

This guide contains references to server configuration settings that JD Edwards EnterpriseOne stores in configuration files (such as jde.ini, jas.ini, jdbj,ini, jdelog.properties, and so on). Beginning with JD Edwards EnterpriseOne Tools Release 8.97, it is highly recommended that you access and manage these settings for the supported server types using the Server Manager program.

Customers must conform to the supported platforms for the release as detailed in the JD Edwards EnterpriseOne minimum technical requirements. In addition, JD Edwards EnterpriseOne may integrate, interface, or work in conjunction with other Oracle products. Refer to the cross-reference material in the Program Documentation at http://oracle.com/contracts/indiex.html for Program prerequisites and version cross-reference documents to assure compatibility of various Oracle products.

See Also:

- *JD Edwards EnterpriseOne Tools Server Manager Guide* on My Oracle Support.
- *JD Edwards EnterpriseOne Tools Reference Guide* on My Oracle Support.
- *JD Edwards EnterpriseOne Tools Transaction Server Reference Guide* on My Oracle Support.
- "JD Edwards EnterpriseOne Tools Business Services Development Overview" in the*JD Edwards EnterpriseOne Tools Business Services Development Guide* .
- "JD Edwards EnterpriseOne Tools Business Services Development Methodology Overview" in the *JD Edwards EnterpriseOne Tools Business Services Development Methodology Guide.*
- *JD Edwards EnterpriseOne Tools Business Services Server Reference Guide* on My Oracle Support.
- JD Edwards EnterpriseOne and Messaging Queue Systems.
- "JD Edwards EnterpriseOne Tools Connectors Overview" in the *JD Edwards EnterpriseOne Tools Connectors Guide*.
- "Table Conversions" in the *JD Edwards EnterpriseOne Tools Development Tools: Data Access Tools Guide*.
- "Understanding OSA" in the *JD Edwards EnterpriseOne Tools Development Tools: Report Printing Administration Technologies Guide*.

Documentation Accessibility

For information about Oracle's commitment to accessibility, visit the Oracle Accessibility Program website at `http://www.oracle.com/us/corporate/accessibility/index.html`.

Access to Oracle Support

Oracle customers have access to electronic support through My Oracle Support. For information, visit `http://www.oracle.com/support/contact.html` or visit `http://www.oracle.com/accessibility/support.html` if you are hearing impaired.

Related Documents

You can access related documents from the JD Edwards EnterpriseOne Release Documentation Overview pages on My Oracle Support. Access the main documentation overview page by searching for the document ID, which is 876932.1, or by using this link:

`https://support.oracle.com/CSP/main/article?cmd=show&type=NOT&id=876932.1`

To navigate to this page from the My Oracle Support home page, click the Knowledge tab, and then click the Tools and Training menu, JD Edwards EnterpriseOne, Welcome Center, Release Information Overview.

This guide contains references to server configuration settings that JD Edwards EnterpriseOne stores in configuration files (such as jde.ini, jas.ini, jdbj.ini, jdelog.properties, and so on). Beginning with the JD Edwards EnterpriseOne Tools Release 8.97, it is highly recommended that you only access and manage these settings for the supported server types using the Server Manager program. See the Server Manager Guide on My Oracle Support.

Conventions

The following text conventions are used in this document:

Convention	Meaning
Bold	Indicates field values.
Italics	Indicates emphasis and JD Edwards EnterpriseOne or other book-length publication titles.
`Monospace`	Indicates a JD Edwards EnterpriseOne program, other code example, or URL.

1 Introduction to JD Edwards EnterpriseOne Tools Interoperability

This chapter contains the following topics:

- Section 1.1, "JD Edwards EnterpriseOne Tools Interoperability Overview"
- Section 1.2, "JD Edwards EnterpriseOne Tools Interoperability Implementation"

1.1 JD Edwards EnterpriseOne Tools Interoperability Overview

Oracle's JD Edwards EnterpriseOne Tools Interoperability is used to send information into or retrieve information from JD Edwards EnterpriseOne. This document identifies the interoperability models and capabilities that JD Edwards EnterpriseOne supports. Depending on which model and capability you use, you must configure the system so that you can send information into or retrieve information from JD Edwards EnterpriseOne. The chapters in this document discuss format and set up requirements.

1.2 JD Edwards EnterpriseOne Tools Interoperability Implementation

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

In the planning phase of your implementation, take advantage of all JD Edwards sources of information, including the installation guides and troubleshooting information.

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

1. Install EnterpriseOne Tools 8.98.

   See the*JD Edwards EnterpriseOne Tools Server Manager Guide* on My Oracle Support.

2. Install JD Edwards EnterpriseOne applications.

   See the *JD Edwards EnterpriseOne Applications Installation Guide* on My Oracle Support.

In addition to the JD Edwards EnterpriseOne Tools and Applications installation guides, install any other EnterpriseOne tools, such as business services and Transaction Server, that are required for the interoperability model that you select.

2 Understanding Interoperability

This chapter contains the following topics:

- Section 2.1, "Interoperability"
- Section 2.2, "Interoperability Features"
- Section 2.3, "Interoperability Models and Capabilities"
- Section 2.4, "Interoperability Model Selection"
- Section 2.5, "Other Industry Standard Support"

2.1 Interoperability

Interoperability is most often associated with software as a way to enable 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 enables business processes to flow from one application to another. Interoperability enables one system to work with another, in near real-time fashion, to share critical business information. Interoperability options become the glue between systems and applications.

2.2 Interoperability Features

Full interoperability among systems makes the flow of data among the systems seamless to the user. Oracle's JD Edwards EnterpriseOne provides a framework to mask the complexity of interoperability with external systems, and to simplify interfacing with third-party packages.

The interoperability solution for JD Edwards EnterpriseOne meets these three important business objectives:

- Flexibility, Options, and Choice

  JD Edwards provides EnterpriseOne-legacy, best-of-breed, customer management, reporting tools, and many other types of applications and information. The developer can make the right choice for the particular environment and needs.

- Investment Preservation

  JD Edwards EnterpriseOne can interface with the existing applications or applications you plan to use in the future. You can use industry standard methods if the existing or new technologies support them, or you can use JD Edwards

EnterpriseOne business logic to create this interoperability. Also, you will benefit from our ongoing upgrades and improvements to that architecture.

- Manageability

  JD Edwards EnterpriseOne is designed to make the interoperability process easily manageable.

2.2.1 Benefits

Interoperability offers these benefits:

- Businesses can bring together applications and systems across an enterprise, irrespective of vendors.
- Collaborations can occur between trading partners to lower the cost of doing business or to 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 as the result of mergers or acquisitions can be quickly incorporated into the enterprise's information technology solution.

The JD Edwards EnterpriseOne interoperability strategy includes a wide range of models and capabilities.

2.3 Interoperability Models and Capabilities

The JD Edwards EnterpriseOne Interoperability matrix provides an overview of interoperability models that are supported by JD Edwards EnterpriseOne. A model is a way for third parties to connect to or access JD Edwards EnterpriseOne. The matrix shows the models, which are further divided into types and into the capabilities that can be used with each model type. The model and model types are listed in the left-hand column. Capabilities, which are ways to send information into or retrieve information from JD Edwards EnterpriseOne, are columns in the matrix. For each model type, you can read across the table to see what capabilities can be used with that model type. JD Edwards provides both interactive and batch capabilities. The capabilities are grouped by inbound, outbound, and batch. An inbound capability is a request for data or a transaction initiated outside of JD Edwards EnterpriseOne. An outbound capability originates inside of JD Edwards EnterpriseOne.

2.3.1 Auditing for Interoperability Transactions

An interoperability transaction can affect a column in a JD Edwards EnterpriseOne table that has been enabled for auditing. When this occurs, JD Edwards EnterpriseOne creates an audit record for the transaction, but the system records only a portion of the audit information, such as the audited column, before and after values, and recorded columns. The audit information will not include a GUID, application ID, workstation name, or IP address, unless you configure the interoperability model to pass this data to the audit record.

See "Configuring Auditing for Interoperability Transactions" in the *JD Edwards EnterpriseOne Tools Auditing Administration Including 21 CFR Part 11 Administration Guide*.

2.3.2 JD Edwards EnterpriseOne Interoperability

This matrix identifies the JD Edwards EnterpriseOne models and the capabilities that each model supports:

Table 2–1 JD Edwards EnterpriseOne Interoperability Models and Capabilities

Model	Model Type	BSFN Calls (In)	XML CallObj, XML List, XML Trans. (In)	Z Trans. (In)	Flat Files (In)	Real-Time and XAPI Events (Out)	Web Services Callout	Generate XML Output (Out)	Flat Files (Out)	Batch (Out)
Business Services Server	Web Services	Y	N	N	N	RTE	Y	Y	N	N
JMS Queue & JMS Topic	J2EE Connectivity	N	N	N	N	RTE Z Events	N	Y	N	N
Web Services Gateway (WSG)	EOne WSG	Y	List*	N	Y	RTE XAPI Z Event	N	Y	Y	N
Connectors	Dynamic Java Connector (Java Connector)	Y	CO, Trans, List*	N	N	RTE XAPI Z Event	N	Y	N	N
Connectors	JCA Resource Adapter	Y	CO, Trans, List*	N	N	N	N	N	N	N
Connectors	COM Connector	Y	CO, Trans, List*	N	N	RTE XAPI Z Event	N	Y	N	N
EOne Messaging Adapters	Adapter for MQ WebSphere	Y	CO, Trans*	Y	N	RTE XAPI Z Event	N	Y	N	Y
EOne Messaging Adapters	Adapter for MSMQ	Y	CO, Trans*	Y	N	RTE XAPI Z Event	N	Y	N	Y
Batch Interfaces	Interface Tables	Y	N	Y	Y	N	N	N	N	Y
Batch Interfaces	EOne EDI	Y	N	Y	Y	N	N	N	Y	Y
Batch Interfaces	Table Conversions	Y	N	Y	Y	N	N	N	Y	Y

* CO, List, and Trans indicate XML CallObj., XML List, XML Trans. from the column heading. These capabilities are XML Call Object, XML List, and XML Transaction. Each of these are discussed in detail in this document.

Table 2–1 (Cont.) JD Edwards EnterpriseOne Interoperability Models and Capabilities

Model	Model Type	BSFN Calls (In)	XML CallObj, XML List, XML Trans. (In)	Z Trans. (In)	Flat Files (In)	Real-Time and XAPI Events (Out)	Web Services Callout	Generate XML Output (Out)	Flat Files (Out)	Batch (Out)
Batch Interfaces	OSA (UBE)	N	N	N	N	N	N	Y	N	Y
APAg/ Integration	APAg/ Integration	N	N	Y	N	RTE	N	Y	N	Y
Open Data Access	Open Data Access (Supports business view and table inquiries)	N/A	N	N	N/A	N/A	N/A	N/A	N/A	N/A

* CO, List, and Trans indicate XML CallObj., XML List, XML Trans. from the column heading. These capabilities are XML Call Object, XML List, and XML Transaction. Each of these are discussed in detail in this document.

See XML CallObject.

See XML Transaction.

See XML List.

2.3.3 Interoperability Capabilities

A capability is a way to transfer information into JD Edwards EnterpriseOne or to retrieve information from JD Edwards EnterpriseOne. The interoperability matrix shows inbound and outbound capabilities and identifies capabilities that are appropriate for batch processing. Inbound capabilities enable you to inquire about data and update (add, change, or delete) data. With inquiry capabilities, you retrieve data for information purposes only. For example, you might want to see prices or availability of an item. You can perform update capabilities on an individual transaction basis or in a batch process, which consists of groups of transactions. An individual transaction update involves updating a single record (for example, adding a purchase order or creating an invoice). Batch processes, which are groups of transactions that typically involve updating multiple records, are usually scheduled to occur at a specific time and are non-interactive. For example, you can upload 10,000 orders to the database at the end of the day or obtain all of the pricing information that has changed and send that information to a web site at the end of the day.

The capabilities available for transferring information into and retrieving information from JD Edwards EnterpriseOne are described briefly in this chapter. Each capability is discussed in further detail in other chapters within this guide.

2.3.3.1 Web Services

Web services provide standardized ways to interoperate between disparate systems. JD Edwards EnterpriseOne provides and consumes web services. As a web service provider, JD Edwards EnterpriseOne exposes web services for consumption by an external system. As a consumer, JD Edwards EnterpriseOne calls an external web service from within the JD Edwards EnterpriseOne business logic layer.

See "JD Edwards EnterpriseOne as a Web Service Provider" in the *JD Edwards EnterpriseOne Tools Business Services Development Guide*.

See "JD Edwards EnterpriseOne as a Web Service Consumer" in the *JD Edwards EnterpriseOne Tools Business Services Development Guide*.

2.3.3.2 J2EE Connectivity

Java 2 Platform, Enterprise Edition (J2EE) provides a distributed, standards-based architecture for implementing highly scalable, reliable, and available e-business applications. JD Edwards EnterpriseOne business services use J2EE connectivity for standards-based messaging, such as JMS Queue and JMS Topic.

2.3.3.3 Business Function Calls

Business function calls are core to JD Edwards EnterpriseOne interoperability. Business functions encapsulate transaction logic to perform specific tasks, such as journal entry transactions, depreciation calculations, and sales order transactions.

JD Edwards EnterpriseOne uses regular business functions and master business functions. A regular business function performs simple tasks, such as tax calculation or account number validation. A master business function (MBF) performs complex tasks and can call several regular business functions to perform those tasks.

See "Understanding Business Functions" in the *JD Edwards EnterpriseOne Tools Development Tools: APIs and Business Functions Guide*.

2.3.3.4 XML

XML provides a flexible, standards-based way of sharing information and moving data among systems. XML enables you to extend enterprise applications and collaborate with business partners and customers. You can use XML CallObject and XML Transaction to update or retrieve JD Edwards EnterpriseOne data. You can use XML List to create an XML data file in the JD Edwards EnterpriseOne system repository and then retrieve the data in small chunks to avoid network traffic. JD Edwards EnterpriseOne output is an XML document.

2.3.3.5 Z Transactions

Z transactions provide inbound capability to JD Edwards EnterpriseOne that enables you to update JD Edwards EnterpriseOne data. JD Edwards EnterpriseOne provides interface tables (Z tables) that support Z transaction capability. You also can create interface tables.

2.3.3.6 Flat Files

Flat files (also known as user-defined formats) are text files that are usually stored on the workstation or server. Flat files do not have relationships defined for them and typically use the Unicode character set. Data in a flat file usually is stored as one continuous string of information. You can use flat files to import or export data from applications that have no other means of interaction. For example, you might want to share information between JD Edwards EnterpriseOne and another system.

2.3.3.7 Events

Events are notifications to third-party applications or end-users that a JD Edwards EnterpriseOne business transaction has occurred. JD Edwards EnterpriseOne supports three kinds of events: Z events, real-time events, and XAPI events. Event data is represented as an XML document.

Z events use interface tables and a batch process to retrieve transaction information and use a Z event generator and the data export subsystem to manage the flow of the outbound data.

Real-time events can be generated from a server or a client. System calls (from a server) and client business function calls (from a client) retrieve transaction information. The transaction information is distributed to subscribers.

XAPI events are real-time events that require a response. A XAPI event is created in the same manner as a real-time event, with additional data structure information for invoking a business function when the response XML document is received.

Event notifications can be sent as web services, using JMS Queue or JMS Topic.

2.3.4 Interoperability Models

JD Edwards EnterpriseOne supports these basic interoperability models:

- Business Services Server
- JMS Queue and JMS Topic
- Connectors
- Messaging Adapters
- Batch Interfaces

These models can be further categorized by type. Each model type supports one or more of the capabilities for sending information into or retrieving information from the JD Edwards EnterpriseOne database. The Interoperability Models and Capabilities matrix identifies the model types and the capabilities that each model type supports.

2.3.4.1 Business Services Server

Business services enable JD Edwards EnterpriseOne to use web services to exchange information with external systems. JD Edwards EnterpriseOne is a web service provider and a web service consumer. The business services server provides a business services development client for developing and testing business services as both a web service provider and a web service consumer.

Some benefits of using business services include:

- Flexibility to interoperate with any web service enabled external system.
- Reduce dependency on embedded third-party products.
- Standards-based integration offerings.
- Simplified integration architecture.
- Increased overall superior ownership experience.

See "Understanding the Business Services Server" in the *JD Edwards EnterpriseOne Tools Business Services Development Guide.*

2.3.4.2 JMS Queue and JMS Topic

JD Edwards EnterpriseOne provides a transaction server that uses Java Message Service (JMS) queues and topics to guarantee event delivery. When an event occurs in JD Edwards EnterpriseOne, the transaction server retrieves the event information and routes it to subscriber JMS queues and topics for each subscriber that has established an active subscription for the event.

Some benefits of using JMS Queue and JMS Topic include:

- Standards-based way of sending messages.
- Guaranteed delivery of events.
- Publish subscribe model supported.
- Point-to-point model supported.

2.3.4.3 Connectors

Connectors are point-to-point, component-based models that enable third-party applications and JD Edwards EnterpriseOne to share logic and data. JD Edwards EnterpriseOne connector architecture includes Java and COM connectors. The connectors accept inbound XML requests and expose business functions for reuse. Output from the connectors is in the form of an XML document. The connectors include:

- Java

  The JD Edwards EnterpriseOne dynamic Java and Java connectors support real-time event processing. Java is a portable language, so you can easily tie JD Edwards EnterpriseOne functionality to Java applications.

- COM

  The JD Edwards EnterpriseOne COM connector solution is fully compliant with the Microsoft component object model. You can easily tie JD Edwards EnterpriseOne functionality to Visual Basic and VC++ applications. The COM connector also supports real-time event processing.

Some benefits of using connectors include:

- Scalability
- Multi-threaded capability
- Concurrent users

See "Getting Started with JD Edwards EnterpriseOne Tools Connectors" in the *JD Edwards EnterpriseOne Tools Connectors Guide*.

2.3.4.4 Messaging Adapters

JD Edwards EnterpriseOne provides messaging support for IBM WebSphere MQ and Microsoft Message Queuing (MSMQ). WebSphere MQ and MSMQ handle message queuing, message delivery, and transaction monitoring. JD Edwards EnterpriseOne uses these messaging systems to handle and pass requests for logic and data between JD Edwards EnterpriseOne and third-party systems.

Some of the benefits of using messaging adapters include:

- Reliable connections
- Guaranteed delivery
- Operations acknowledgement

  See JD Edwards EnterpriseOne and Messaging Queue Systems.

2.3.4.5 Batch Interfaces

Batch implies processing multiple transactions at the same time and usually involves movement of bulk information. Batch processing is often scheduled and is non-interactive. JD Edwards EnterpriseOne provides several model types for batch processing, and each model type has one or more capabilities that enable you to access JD Edwards EnterpriseOne data. The model types include:

- Interface tables
- Electronic Data Exchange
- Table conversions
- Output Stream Access
- APAg/Integration
- Open Data Access

2.3.4.6 Interface Tables

Interface tables provide point-to-point interoperability solutions for importing and exporting data. Interface tables are also called Z tables. Interface tables are working files into which you place transaction information to be processed into or out of JD Edwards EnterpriseOne. In addition to the interface tables provided by JD Edwards EnterpriseOne, you can build interface tables. If you use interfaces tables to update JD Edwards EnterpriseOne data, the format of the data must be presented in the format defined by JD Edwards EnterpriseOne. If you use interface tables to retrieve JD Edwards EnterpriseOne data, you use a batch process that extracts the data from the applications tables.

Some of the benefits of using interface tables include:

- Defined data structure
- Identifiable fields
- Customizable interface tables

2.3.4.7 EDI

Electronic Data Interchange (EDI) provides a point-to-point interoperability solution for importing and exporting data. 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 that use EDI must have translator software to convert the data from the EDI standard format to the format of their computer system.

The JD Edwards EnterpriseOne Data Interface for Electronic Data Interchange system acts as an interface between the JD Edwards EnterpriseOne system data and the translator software. In addition to exchanging EDI data, this data interface also can 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 include:

- 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.

See *JD Edwards EnterpriseOne Data Interface for Electronic Data Interchange 9.0 Implementation Guide*.

2.3.4.8 Table Conversion

Table conversion provides a point-to-point interoperability solution for importing and exporting data. Table conversion is a special form of Universal Batch Engine (UBE) that enables you to do high-speed manipulation of data in tables. JD Edwards EnterpriseOne has a table conversion utility that you can use to gather, format, import, and export data. The table conversion tool enables you to transfer and copy data. You can also delete records from tables. Table conversion enables you to use a non-JD Edwards EnterpriseOne table to process, call direct business functions, and give an output. For example, you might want to run a UBE that reads from a JD Edwards EnterpriseOne master file to populate a non-JD Edwards EnterpriseOne table.

The table conversion utility can make use of any JD Edwards EnterpriseOne table, business view, and text file, or any table that is not a JD Edwards EnterpriseOne table but resides in a database that is supported by JD Edwards EnterpriseOne, such as Oracle, Access, IBM i, or SQL Server. These non-JD Edwards EnterpriseOne tables are commonly referred to as foreign tables.

See "Understanding Table Conversion" in the *JD Edwards EnterpriseOne Tools Development Tools: Data Access Tools Guide*.

2.3.4.9 OSA

OSA (Output Stream Access) provides a point-to-point interoperability solution for exporting data from UBEs. OSA enables you to set up an interface for JD Edwards EnterpriseOne to pass data to another software package, such as Microsoft Excel, for processing.

The benefits for using OSA include:

- The elimination of manually formatting output.
- The processing power of the target software program.

  See "Working with Output Stream Access" in the *JD Edwards EnterpriseOne Tools Development Tools: Report Printing Administration Technologies Guide*.

2.3.4.10 APAg/Integration

The JD Edwards EnterpriseOne Advanced Planning Agent (APAg) is a tool for batch extraction, transforming, and loading enterprise data. APAg supports access to data sources in the form of relational databases, flat file format, and other data or message encoding such as XML. APAg also moves data from one place to another and initiates tasks related to the movement of the data.

Benefits of using the APAg tool include:

- Ability to copy massive amounts of table data.
- Ability to efficiently and effectively handle initial data loads.

See "JD Edwards Supply Chain Planning" in the *Advanced Planning Agent Guide*.

2.3.4.11 ODA

ODA (Open Data Access) provides the capability for you to extract JD Edwards EnterpriseOne data (using SQL statements) so that you can summarize information and generate reports. You can use ODA with any of these desktop applications:

- Microsoft Query

- Microsoft Access
- Microsoft Excel
- ODBCTEST
- Crystal Reports
- Microsoft Analysis Service

ODA sits between the front-end query and reporting applications and the JD Edwards EnterpriseOne-configured ODBC drivers.

The JD Edwards EnterpriseOne 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, media object, 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 application.

Some of the benefits of using ODA include:

- Read-only access to all JD Edwards EnterpriseOne data, including the entire data dictionary.
- Use of the same security rules that you established for JD Edwards EnterpriseOne.
- Ability to extract JD Edwards EnterpriseOne data easily.

  See Understanding Open Data Access.

2.4 Interoperability Model Selection

Select an interoperability model based on the business needs. This matrix can help you determine which interoperability model best supports the interoperability requirements.

Model	Model Type	Platforms (Windows, UNIX, IBM i)	Integration Model	Best Fit Programming Languages	Critical Technical Skills for Creating Inbound Transaction	Critical Technical Skills for Creating Outbound Transactions
Business Services Server	Web Services	Oracle Application Server or WebSphere Application Server	Web Services	Java	Java, Web Services, Database Ops, Business Functions, XML	Java, Web Services, XML
JMS Queue / JMS Topic	J2EE Connectivity	Oracle Application Server or WebSphere Application Server	Web Services	Java	Java, Web Services, Database Ops, Business Functions, XML	Java, Web Services, Database Ops, Business Functions, XML
Web Services Gateway (WSG)	EnterpriseOne WSG	SUN, AIX, Windows	Broker	Java	WSG Toolset	WSG Toolset, Real-Time Events, XAPI Events

Model	Model Type	Platforms (Windows, UNIX, IBM i)	Integration Model	Best Fit Programming Languages	Critical Technical Skills for Creating Inbound Transaction	Critical Technical Skills for Creating Outbound Transactions
Connectors	Java Connector	All	Point-to-Point	Java	Java APIs, GenJava	Real-Time Events, XAPI Events
Connectors	COM Connector	Windows	Point-to-Point	C/C++/VB	COM, GenCOM	Real-Time Events, XAPI Events
Messaging Adapters	Adapter for MQ WebSphere	All	Integration Server	HTML, C/C++, Java	MQ WebSphere, XML	Real-time Events, Z-Tables, Subsystem Processing (includes R00460, Data Export Controls, and so on)
Messaging Adapters	Adapter for MSMQ	Windows	Integration Server	C/C++	MSMQ, XML	Real-Time Events, Z-Tables, Subsystem Processing (includes R00460, Data Export Controls, and so on)
Batch Interfaces	Interface Tables	All	Point-to-Point	Any	Z-Tables, UBEs	Custom Code
Batch Interfaces	EnterpriseOne EDI	All	Point-to-Point	Any, Flat Files	Z-Tables, UBEs	Custom Code
Batch Interfaces	Table Conversions	All	Point-to-Point	TC	Table Conversions	Table Conversion Director/RDA
Batch Interfaces	OSA (UBE)	All	Point-to-Point	HTML, C/C++	NA	RDA, Custom Code
APAg/ Integration	APAg/ Integration	UNIX, Windows	Point-to-Point	Own Language	APAg Tool, Z Tables	APAg Tool, Z-Tables
Open Data Access	Open Data Access	All	Point-to-Point	VB	Custom code or third-party application (queries only)	Custom code or third-party application (queries only)

2.5 Other Industry Standard Support

JD Edwards EnterpriseOne has a media object function that supports other industry standard functions, such as:

- Object Linking and Embedding (OLE) for the exchange of different data types.
- Dynamic Data Exchange (DDE) for static and dynamic links across applications.
- Binary Large Object (BLOB) for media object attachments within applications.

- Extended Messaging API (MAPI) for message exchange across differing mail and groupware applications.

See Also:

- "Understanding Media Object Attachments" in the *JD Edwards EnterpriseOne Tools Foundation Guide*.
- "Understanding Messages and Queues" in the *JD Edwards EnterpriseOne Tools Foundation Guide*.

3 Understanding Integrations in a SOA Environment

This chapter contains the following topics:

- Section 3.1, "JD Edwards Enterprise Integrations in a SOA Environment"
- Section 3.2, "Business Services Architecture"
- Section 3.3, "Environments"
- Section 3.4, "Integration Patterns"

3.1 JD Edwards Enterprise Integrations in a SOA Environment

As systems evolve, Service Oriented Architecture (SOA) environments are instrumental for providing a standards-based approach for interoperability between disparate systems. In a SOA environment, web services provide a common interface between systems. JD Edwards EnterpriseOne provides and consumes web services in a SOA environment by leveraging business services. JD Edwards EnterpriseOne also supports event notification in a SOA environment using JMS Queue and JMS Topic.

3.1.1 Web Service Provider

As a web service provider, JD Edwards EnterpriseOne exposes web services for consumption by external systems. JD Edwards EnterpriseOne web services call business services. Business services perform a specific business process. Multiple Java classes are used to perform the requested business process. The web service is generated from a Java class called a published business service class. The methods of the published business service class receive and return data through payload classes called value objects. Within each method, internal business service and value object classes are used to access existing logic and data in JD Edwards EnterpriseOne. The business processes exposed through the published business service class can be accessed from an external system using a web service call or from other published business service classes.

3.1.2 Web Service Consumer

As a web service consumer, JD Edwards EnterpriseOne calls an external web service from within the JD Edwards EnterpriseOne business logic layer. An action that uses a business function occurs in JD Edwards EnterpriseOne. The business function calls a business service. The business service calls the external web service. A web service proxy provides end points and security information for the external web service. The results of the call are returned to the published business service that is provided in the web service proxy. The published business service calls the business service method, which passes the result to the business function. JD Edwards EnterpriseOne can also consume web services using HTTP instead of the business services server.

See "Understanding Business Services Development in the *JD Edwards EnterpriseOne Tools Business Services Development Guide*.

3.1.3 Event Notification

JD Edwards EnterpriseOne sends event notifications as JMS messages through JMS Queue and JMS Topic. The transaction server is the primary business event system for publishing guaranteed event notifications. When a transaction occurs in JD Edwards EnterpriseOne, the transaction server retrieves the data based on event configuration, converts the data to a properly formatted XML document, and routes the event to the JMS Queue or JMS Topic subscriber.

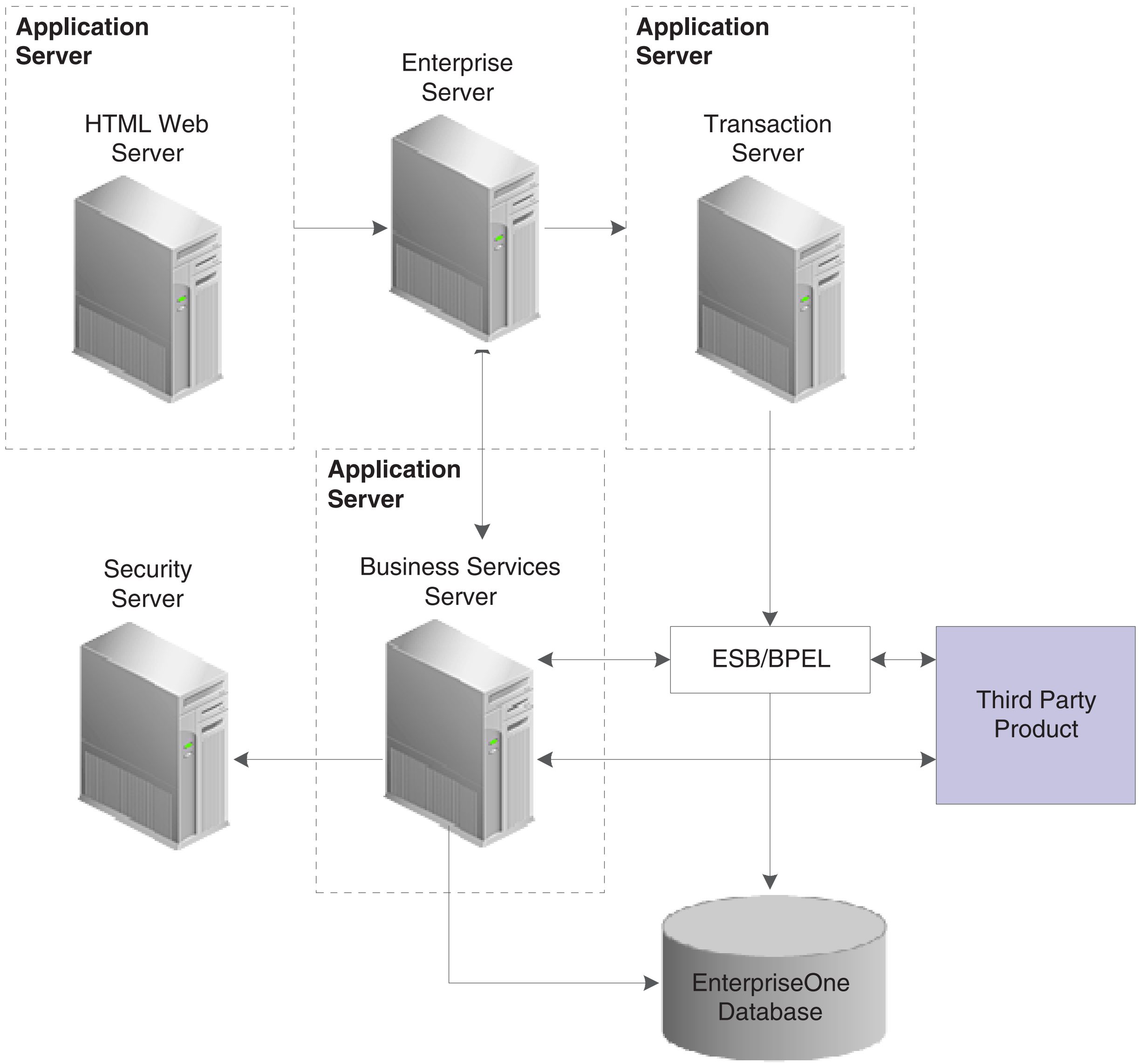
See Also:

- Understanding Guaranteed Events.
- "Understanding Business Services Development" in the *JD Edwards EnterpriseOne Tools Business Services Development Guide*.

3.2 Business Services Architecture

The following diagram illustrates the architecture for JD Edwards EnterpriseOne web services and business services:

Figure 3–1 Architecture for JD Edwards EnterpriseOne web services and business services



This table discusses the servers and systems depicted in the diagram:

System	Description
Application Server	Runs the business services server, the transaction server, and the HTML Web server. The application server can be an Oracle Application Server or a WebSphere Application Server.
HTML Web Server	Runs JD Edwards EnterpriseOne interactive applications. Communicates with the enterprise server to run business functions.
Enterprise Server	Runs business functions that generate request/reply messaging events.
Transaction Server	Transports the XML message generated from the request/reply messaging API to the receiving systems using JMS Queue and JMS Topic.
Security Server	Provides authentication for JD Edwards EnterpriseOne components.

System	Description
Business Services Server	Hosts the business service Java programs that communicate with JD Edwards EnterpriseOne. Provides a business services development client for developing and testing services as both a web service provider and a web service consumer.
Orchestration System	Used for SOA orchestration, for example, Oracle BPEL-PM and Oracle ESB.
Database Server	Hosts tables.

3.3 Environments

JD Edwards EnterpriseOne provides a business services development client for developing and testing business services as both a web service provider and a web service consumer.

3.4 Integration Patterns

JD Edwards EnterpriseOne supports the following integration patterns for interoperating with other Oracle applications and third-party applications or systems:

- JD Edwards EnterpriseOne as a web service provider – synchronous request/reply.
- JD Edwards EnterpriseOne as a web service provider – asynchronous notification.
- JD Edwards EnterpriseOne as a web service provider – asynchronous request/reply.
- JD Edwards EnterpriseOne as a web service consumer – notification.
- JD Edwards EnterpriseOne as a web service consumer – synchronous web service request/reply.
- JD Edwards EnterpriseOne as a service consumer – asynchronous HTTP request/response.
- JD Edwards EnterpriseOne as a service consumer – synchronous HTTP request/response.
- JD Edwards EnterpriseOne as a web service consumer – asynchronous web service.

These patterns are typically used for point-to-point integrations with individual third-party systems.

3.4.1 JD Edwards EnterpriseOne as a Web Service Provider - Synchronous Request/Reply

JD Edwards EnterpriseOne supports two methods for processing the web service provider synchronous request/reply pattern. The most frequently used model is to expose a web service that accesses the JD Edwards EnterpriseOne data through a set of business function calls.

This pattern uses these systems:

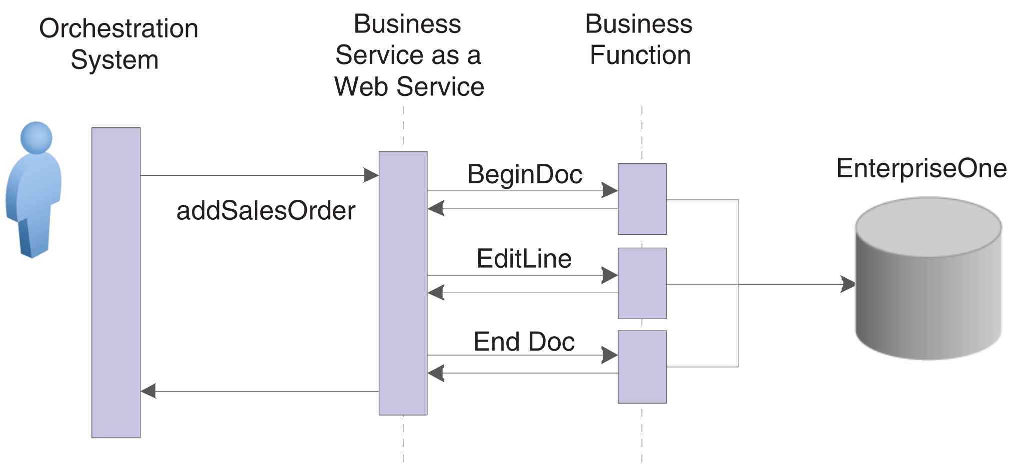
- Orchestration system
- Business services server
- Enterprise server

- Database server

The orchestration system calls a JD Edwards EnterpriseOne web service. The web service calls a business service. The business service calls a business function. The business function performs the task that updates the JD Edwards EnterpriseOne database.

This diagram shows this model:

Figure 3–2 JD Edwards EnterpriseOne as a web service provider - synchronous request/reply using business function calls



The other method uses JDBj to perform direct data access to the JD Edwards EnterpriseOne database.

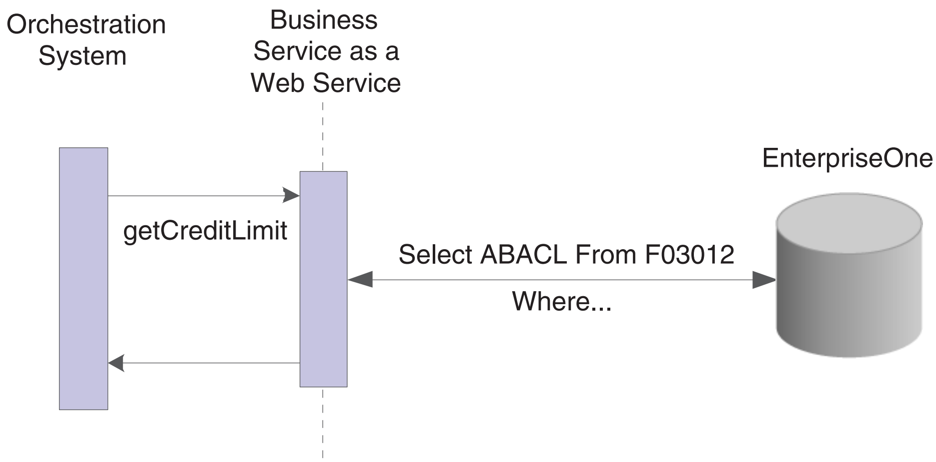
This pattern uses these systems:

- Orchestration system
- Business services server
- Database server

The orchestration system calls a JD Edwards EnterpriseOne web service. The web service calls a business service. The business service makes a database operation call that updates the JD Edwards EnterpriseOne database.

This diagram illustrates this model:

Figure 3–3 JD Edwards EnterpriseOne as a web service provider - synchronous request/reply using JDBj



3.4.2 JD Edwards EnterpriseOne as a Web Service Provider - Asynchronous Notification

JD Edwards EnterpriseOne supports two methods for processing the web service provider asynchronous notification pattern. The most frequently used method is to expose a web service that accesses the JD Edwards EnterpriseOne data through a set of business function calls.

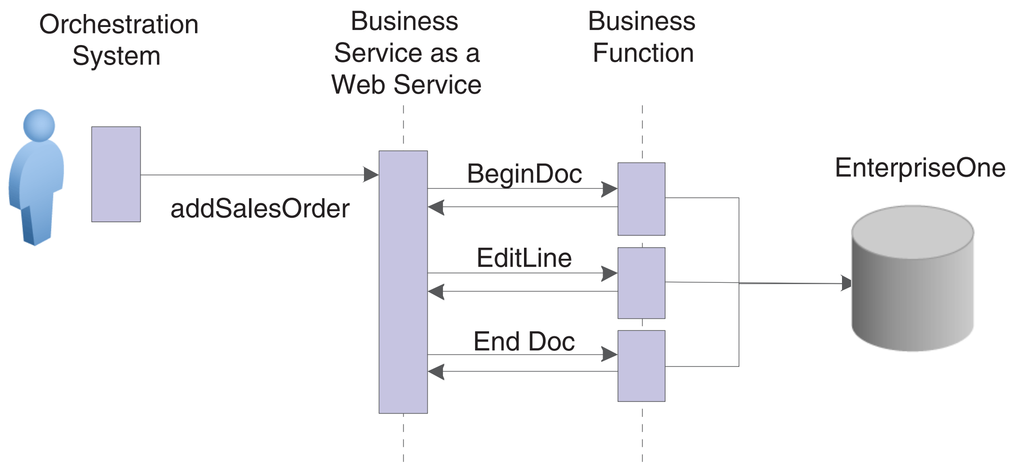
This pattern uses these systems:

- Orchestration system
- Business services server
- Enterprise server
- Database server

The orchestration system calls a JD Edwards EnterpriseOne web service. The web service calls a business service. The business service calls a business function. The business function performs the task that updates the JD Edwards EnterpriseOne database.

This diagram illustrates this model:

Figure 3–4 JD Edwards EnterpriseOne as a web service provider - asynchronous notification using business function calls



The other method uses JDBj to perform direct data access to the JD Edwards EnterpriseOne database.

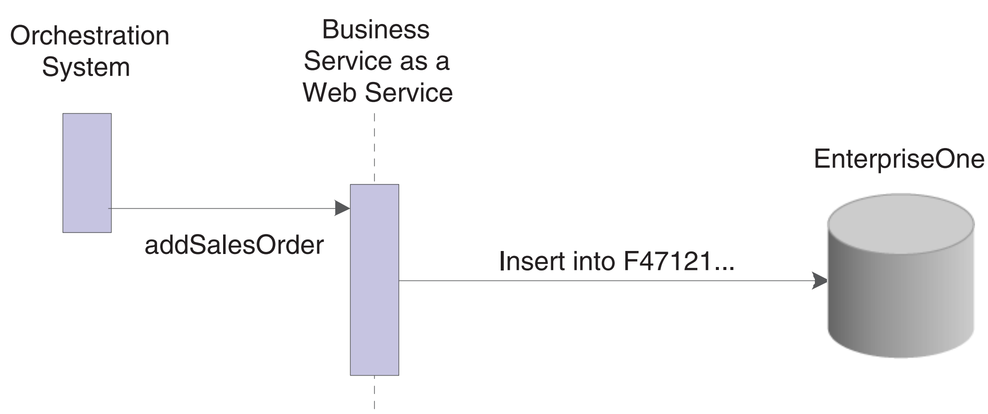
This pattern uses these systems:

- Orchestration system
- Business services server
- Database server

The orchestration system calls a JD Edwards EnterpriseOne web service. The web service calls a business service. The business service makes a database operation call that updates the JD Edwards EnterpriseOne database.

This diagram illustrates this model:

Figure 3–5 JD Edwards EnterpriseOne as a web service provider - asynchronous notification using JDBj



3.4.3 JD Edwards EnterpriseOne as a Web Service Provider - Asynchronous Request/Reply

JD Edwards EnterpriseOne supports two methods for processing the web service provider asynchronous request/reply pattern. The most frequently used method is to expose a web service that accesses the JD Edwards EnterpriseOne data through a set of business function calls.

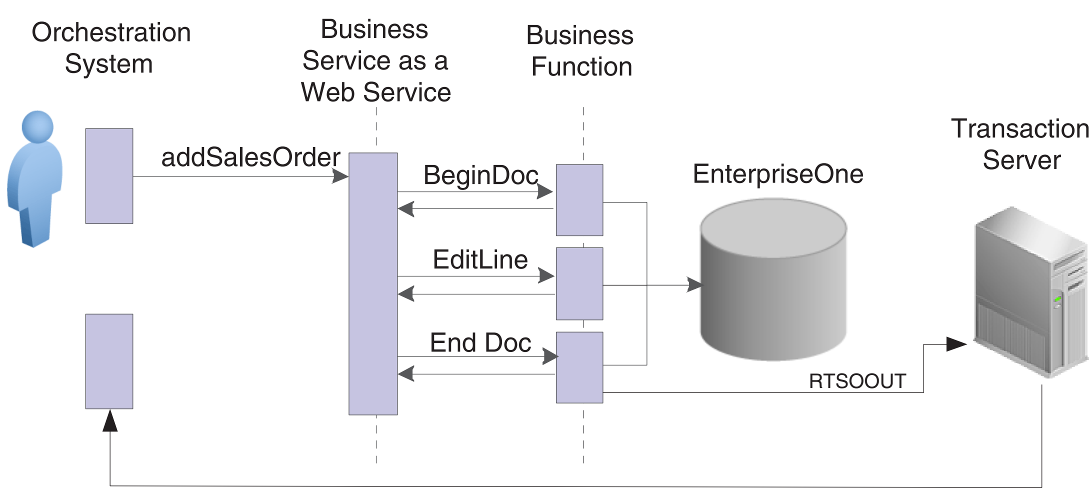
This pattern uses these systems:

- Orchestration system
- Business services server
- Enterprise server
- Database server
- Transaction server

The orchestration system calls a JD Edwards EnterpriseOne web service. The web service calls a business service. The business service calls a business function. The business function performs the task that updates the JD Edwards EnterpriseOne database. The EnterpriseOne application notifies the transaction server that an update has occurred. The transaction server retrieves the information and creates an event (outbound notification) and places the event in JMS Queue or JMS Topic for the orchestration system to send to the subscriber. The reply is received through the orchestration system and returned to JD Edwards EnterpriseOne as an XML document through the transaction server.

This diagram illustrates this model:

Figure 3–6 JD Edwards EnterpriseOne as a web service provider - asynchronous request/reply using business function calls



3.4.4 JD Edwards EnterpriseOne as a Web Service Consumer - Notification

JD Edwards EnterpriseOne supports two methods for processing the web service consumer asynchronous notification pattern. The most frequently used method is to publish a real-time event using the transaction server.

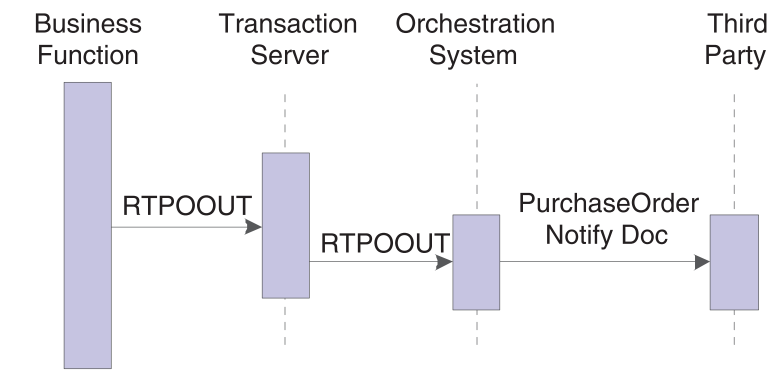
This pattern uses these systems:

- Enterprise server
- Transaction server
- Orchestration system

A business function performs a task that updates the JD Edwards EnterpriseOne database. The Call Object kernel notifies the transaction server. The transaction server retrieves the data and creates an event in the form of an XML document and places the event in JMS Queue or JMS Topic for the orchestration system to process. The orchestration system retrieves the XML document and sends it to the third-party system. The data mapping between the request and reply is provided by the cross-reference correlation utility in the orchestration system.

This diagram illustrates this model:

Figure 3–7 JD Edwards EnterpriseOne as a web service consumer — asynchronous notification using real-time events



The other method uses Z-tables to send information to third-party systems.

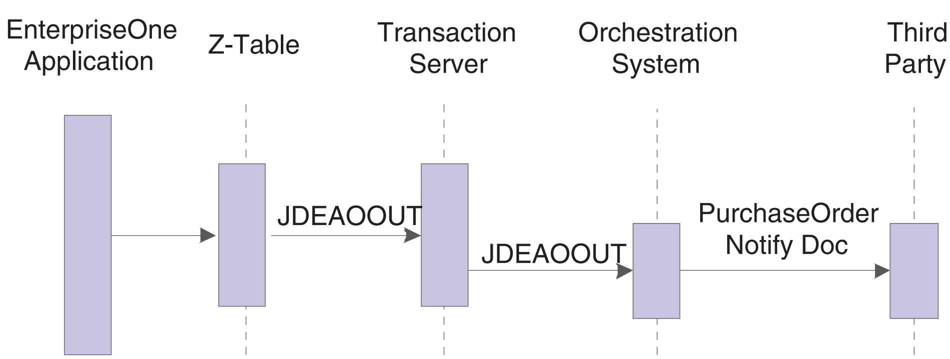
This pattern uses these systems:

- Enterprise server
- Transaction server
- Orchestration system

An update is made to a JD Edwards EnterpriseOne application. The application has processing options that load data into a specified Z-table. The system is then configured to publish the Z-table record using the transaction server.

This diagram illustrates this model:

Figure 3–8 JD Edwards EnterpriseOne as a web service consumer – outbound notification using Z-tables



3.4.5 JD Edwards EnterpriseOne as a Web Service Consumer – Synchronous Web Service Request/Reply

JD Edwards EnterpriseOne supports using a web service for processing the synchronous request/reply pattern. This method uses a JD Edwards business service to call an external web service.

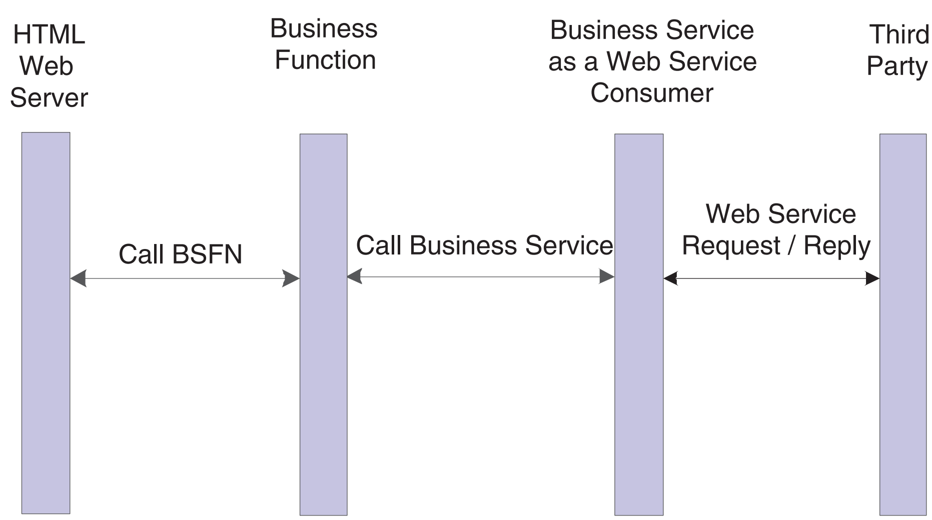
This pattern uses these systems:

- HTML web server
- Enterprise server
- Business services server

A request for information from a third-party system is made through the JD Edwards EnterpriseOne HTML web client. This request invokes a business function. The business function calls a business service. The business service calls an external web service. A web service proxy provides end points and security information for calling the external web service. The results of the call are returned to a JD Edwards EnterpriseOne published business service, which calls a business service to pass the results to the business function, which then processes the information for the HTML web client.

This diagram illustrates this model:

Figure 3–9 JD Edwards EnterpriseOne as a web service consumer – synchronous web service request/reply



3.4.6 JD Edwards EnterpriseOne as a Service Consumer – Asynchronous HTTP Request/Response

JD Edwards EnterpriseOne supports using HTTP POST for processing an asynchronous HTTP request/response pattern. This method uses HTTP POST as the request and expects an HTTP callback. In this pattern, the web server client continues to process other information while waiting for the response.

This pattern uses these systems:

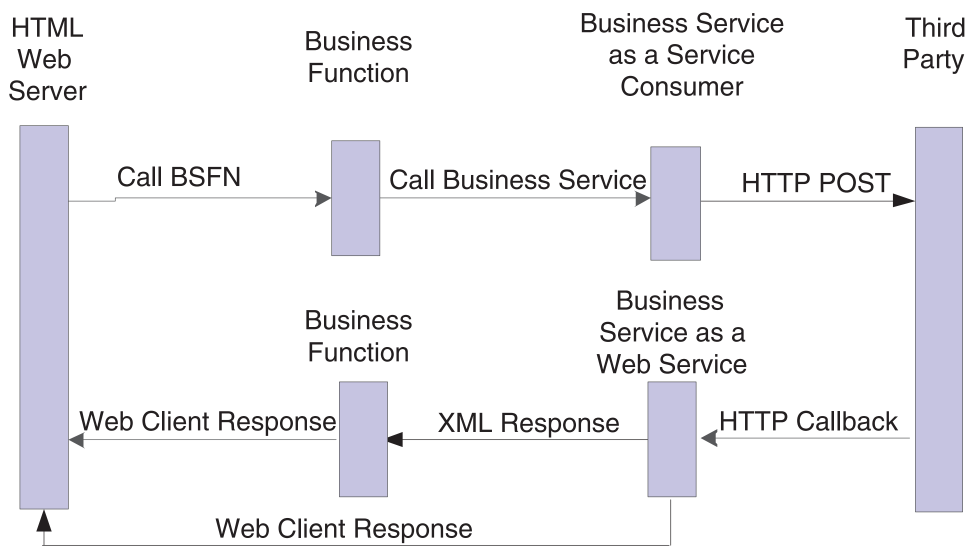
- HTML web server
- Enterprise server

- Business services server

A request for information from a third-party system is made through the JD Edwards EnterpriseOne HTML web client. This request invokes a JD Edwards EnterpriseOne business function. The business function calls a JD Edwards EnterpriseOne business service. The business service contains the request and callback information for the third-party system. The third-party system uses the callback information to send a response that is in XML format to a JD Edwards EnterpriseOne published business service. The published business service can send the response to the business function, and the business function sends the response to the HTML web client. The published business service can also send the response to the HTML web client directly.

This diagram illustrates this model:

Figure 3–10 JD Edwards EnterpriseOne as a service consumer – asynchronous HTTP request/response



3.4.7 JD Edwards EnterpriseOne as a Service Consumer – Synchronous HTTP Request/Response

JD Edwards EnterpriseOne supports using HTTP POST for processing a synchronous HTTP request/response pattern. This method uses HTTP POST as the request and waits for the response from the third-party system.

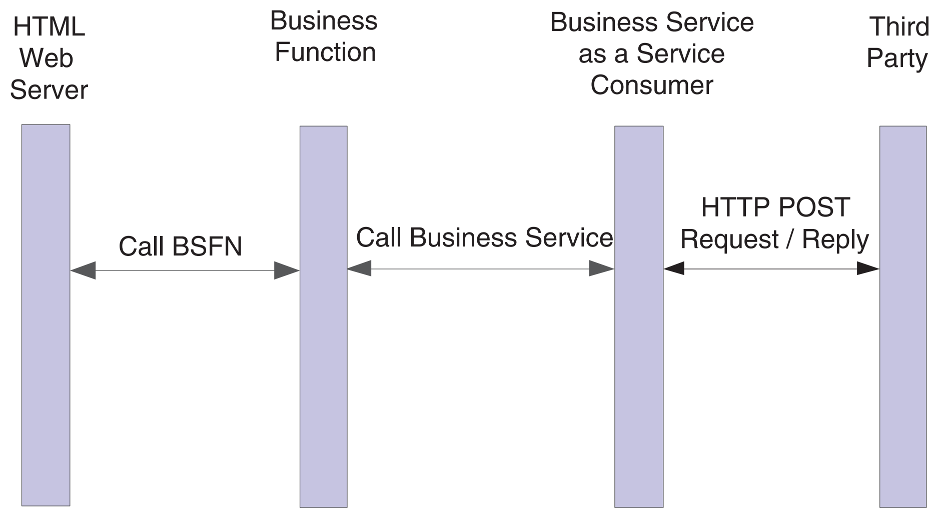
This pattern uses these systems:

- EnterpriseOne HTML web server
- EnterpriseOne server
- Business services server

A request for information from a third-party system is made through the JD Edwards EnterpriseOne HTML web client. This request invokes a JD Edwards EnterpriseOne business function. The business function calls a JD Edwards EnterpriseOne business service. The business service calls the third-party and receives a reply in XML format. The business service sends the response to the business function, and the business function sends the response to the HTML web client.

This diagram illustrates this model:

Figure 3–11 JD Edwards EnterpriseOne as a web service consumer – synchronous HTTP request/response



3.4.8 JD Edwards EnterpriseOne as a Web Service Consumer – Asynchronous Web Service

You can initiate an asynchronous request by leveraging either JD Edwards EnterpriseOne as a Web Service Consumer – Notification or JD Edwards EnterpriseOne as a Web Service Consumer – Synchronous Web Service Request/Reply, and then use JD Edwards EnterpriseOne as a Web Service Provider to handle the response back into EnterpriseOne. JD Edwards EnterpriseOne does not provide any specific feature such as correlation or web services addressing to support calling a web service for processing the asynchronous request/reply pattern. If you use this pattern, you must manage correlation data using application data or payload such as an order number.

4 Using Business Function Calls

This chapter contains the following topics:

- Section 4.1, "Understanding Business Functions"
- Section 4.2, "Reviewing API and Business Function Documentation"
- Section 4.3, "Creating Business Function Documentation"
- Section 4.4, "Finding Business Functions"

4.1 Understanding Business Functions

A business function is an encapsulated set of business rules and logic that can be reused by multiple applications. Business functions provide a common way to access the JD Edwards EnterpriseOne database. A business function accomplishes a specific task. Master business functions provide the logic and database calls necessary to extend, edit, and commit the full transaction to the database. Third-party applications can use master business functions for full JD Edwards EnterpriseOne functionality, data validation, security, and data integrity.

You can use master business functions to update master files (such as Address Book Master and Item Master) or to update transaction files (such as sales orders and purchase orders). Generally, master file master business functions, which access tables, are simpler than transaction file master business functions, which are specific to a program. Transaction master business functions provide a common set of functions that contain all of the necessary default values and editing for a transaction file. Transaction master business functions contain logic that ensures the integrity of the transaction being inserted, updated, or deleted from the database.

For interoperability, you can use master file master business functions instead of table input and output. Using master business functions enables you to perform updates to related tables using the master business function instead of table event rules. In this case, the system does not use multiple records; instead, all edits and actions are performed with one call.

Business functions are core for interoperability with JD Edwards EnterpriseOne. If you build custom integrations to interoperate with JD Edwards EnterpriseOne, you must know which business functions to call and how to call those business functions. You can use existing business functions, modify existing business functions, or create custom business functions. If you are creating a custom business function, JD Edwards suggests that you find an existing business function that is similar to what you want to accomplish and use the existing business function as a model.

Note: When an update or an Electronic Software Update (ESU) affects business functions, you might be required to modify the custom integration.

See "Understanding Business Functions" in the *JD Edwards EnterpriseOne Tools Development Tools: APIs and Business Functions Guide*.

4.2 Reviewing API and Business Function Documentation

You can use JD Edwards EnterpriseOne business functions and APIs in custom integrations. Business functions groupings are:

- Master Business Functions

  A collection of business functions that provide the logic and database calls that are necessary to extend, edit, and commit the full transaction to the database. The design of master business functions enables them to be called asynchronously and to send coded error messages back to calling applications.

- Major Business Functions

  Components that encapsulate reusable logic common to many applications, such as date editing routines and common multicurrency functions.

- Minor Business Functions

  Components that perform complex logic for a specific instance or single application. Minor business functions are used in JD Edwards EnterpriseOne for processing that cannot be accomplished efficiently in event rules or for logic that might be required in multiple places within a single application.

4.3 Creating Business Function Documentation

Business function documentation explains what individual business functions do and how to use each business function. You can generate information for all business functions, groups of business functions, or individual business functions. The documentation for a business function includes information such as:

- Purpose.
- Parameters (the data structure).
- Explanation of individual parameter that indicate the input/output required and an explanation of return values.
- Related tables (which tables are accessed).
- Related business functions (business functions that are called from within the functions itself).
- Special handling instructions.

See "Understanding Business Function Documentation" in the *JD Edwards EnterpriseOne Tools Development Tools: APIs and Business Functions Guide* .

4.4 Finding Business Functions

If you can find a JD Edwards EnterpriseOne application that is similar to what you need to do, you can use that application as a model. The JD Edwards EnterpriseOne

Cross Application Development Tools menu (GH902) provides several tools that you can use to determine what business functions a JD Edwards EnterpriseOne application uses and how the business function is used in the application. From the Cross Application Development Tools menu, you can access:

- Object Management Workbench
- Cross Reference Facility
- Debug Application

4.4.1 Using the Object Management Workbench

You can use the Object Management Workbench (OMW) to search for the business function object and then review the C code.

See "Understanding Objects" in the *JD Edwards EnterpriseOne Tools Object Management Workbench Guide*.

See "JD Edwards EnterpriseOne OMW Projects" in the *JD Edwards EnterpriseOne Tools Object Management Workbench Guide*.

4.4.2 Using the Cross Reference Facility

You can use the Cross Reference Facility to identify each instance for which a business function is used. The Cross Reference program (P980011) is on the Cross Application Development Tools menu (GH902).

See "Understanding the Cross Reference Facility" in the *JD Edwards EnterpriseOne Tools Development Tools: Data Access Tools Guide*.

4.4.3 Using the Debug Application

Another option that you might consider for understanding a JD Edwards EnterpriseOne application is to run a JD Edwards EnterpriseOne debugger. You can run the Event Rules Debugger to obtain named event rule and table event rule information for a JD Edwards EnterpriseOne application. You can use Microsoft Visual C++ to debug business functions that are written in C. You can use these two tools together.

See "Understanding the Event Rules Debugger" in the *JD Edwards EnterpriseOne Tools Development Tools: Event Rules Guide*.

See "Understanding the Visual C++ Debugger" in the *JD Edwards EnterpriseOne Tools Development Tools: APIs and Business Functions Guide*.

5 Understanding XML

This chapter contains the following topics:

- Section 5.1, "XML and JD Edwards EnterpriseOne"
- Section 5.2, "XML JAR Files"
- Section 5.3, "XML Document Format"
- Section 5.4, "XML Standards"
- Section 5.5, "System Environment Configuration"
- Section 5.6, "XML Kernel Troubleshooting"

5.1 XML and JD Edwards EnterpriseOne

Oracle's JD Edwards EnterpriseOne XML solution supports well-formed XML documents. The XML solution supports both UTF-8 and UTF-16 Unicode standards for receiving information into JD Edwards EnterpriseOne. The XML solution supports UTF-8 Unicode standard for sending information from JD Edwards EnterpriseOne. The JD Edwards EnterpriseOne XML solution includes:

XML Solution	Description
XML Transformation System (XTS)	Transforms an XML document that is not in the JD Edwards EnterpriseOne format into an XML document that can be processed by JD Edwards EnterpriseOne, and then transforms the response back to the original XML format.
XMLDispatch	Provides a single point of entry for all XML documents coming into JD Edwards EnterpriseOne and for responses.
XML CallObject	Enables you to call business functions.
XML Transaction	Enables you to use a predefined transaction type (such as JDEOPIN) to send information to or request information from JD Edwards EnterpriseOne. XML transaction uses interface table functionality.
XML List Kernel	Enables you to request and receive JD Edwards EnterpriseOne database information in chunks.
XML Service Kernel	Enables you to request events from one JD Edwards EnterpriseOne system and receive a response from another JD Edwards EnterpriseOne system.

Some of the benefits of using XML include:

- Scalable XML models that enable you to open multiple connections.
- Ability to use JD Edwards EnterpriseOne messaging adapters, providing a reliable connection and acknowledging operations.
- Exposure of business functions and interface tables.
- Ability to aggregate business function calls into one document, which reduces network traffic.
- Ability to manage session creation, validation, and tracking.

If you can create XML documents on the interoperability server, you can use XML for the interoperability solution.

5.2 XML JAR Files

For XML interoperability to function properly, you must add the following jar files to the classpath on the machine that is running XML requests:

- Base_JAR.jar
- commons-httpclient-3.0.jar
- commons-logging.jar
- JdeNet.JAR.jar
- jmxremote_optional.jar
- jmxri.jar
- log4j.jar
- ManagementAgent_JAR.jar
- System_JAR.jar
- xerces.jar
- xmlparserv2.jar

You can find these jar files in the <JD Edwards EnterpriseOne Windows client installation directory>system\classes folder.

5.3 XML Document Format

This section provides an overview of formatting XML documents for JD Edwards EnterpriseOne and discusses these elements:

- Type Element
- Establish Session
- Expire Session
- Terminate Session
- Explicit Transaction
- Implicit Transaction
- Prepare/Commit/Rollback
- Terminate Session

5.3.1 Formatting XML Documents

When you send an XML document to JD Edwards EnterpriseOne for processing, the document must be in the XML format that is defined by JD Edwards EnterpriseOne. After the document reaches the JD Edwards EnterpriseOne server, the system processes the document based on the document type. All XML documents must contain these elements:

- One of these types:
  - jdeRequestType
  - jdeResponseType
- Establish Session
- Expire Session
- Terminate Session

In addition, you can use these optional elements:

- Explicit Transaction
- Implicit Transaction
- Prepare/Commit/Rollback

5.3.2 Type Element

The type element, which can be jdeRequest or jdeResponse, is the root element for all request documents coming into the XML infrastructure. This element contains basic information about the execution environment. These attributes form the jdeRequest and jdeResponse type element:

Attribute	Description
Type	Specifies the type of XML document request. Depending on the operation to be performed, the jdeRequest type can be one of the these: ■ Callmethod ■ List ■ Trans ■ xapicallmethod The jdeResponse type indicates an XML document coming from another JD Edwards EnterpriseOne system. The operation for jdeResponse is realTimeEvent. **Note:** The xapicallmethod and realTimeEvent types are discussed in the Events section of this document.
User	Specifies the user name for user identification and validation.
Pwd	Specifies the user password for user identification and validation.
Role	Specifies the user role. If left blank the default value is *ALL
Environment	Specifies the system environment.
Session	Specifies the session ID. This attribute is optional.
Sessionidle	Specifies the session time-out time. This attribute is optional.

5.3.3 Establish Session

You establish a session by setting the session attribute of the standard jdeRequest element. When the session attribute is an empty string, a new session is started. On the server, the SessionManager singleton class creates a new instance of a session object given the user name, password, and environment name. The session can be reused before it expires to avoid the overhead of session initialization. You can specify the session ID in the session attribute for an already established session in an earlier request.

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

> **Note:** If you do not want to start a new session, then remove the session=' ' tag. This example is for starting a new session.

5.3.4 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 value is 30 minutes. The session idle time is defined in the XML document.

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

5.3.5 Explicit Transaction

Explicit database transactions are supported by another element, the startTransaction tag. The startTransaction tag specifies whether transactions are to be manually or automatically committed. The startTransaction tag element is an empty element, which means that all of the information is in the attributes.

```
<?xml version='1.0'?>
<jdeRequest type='callmethod' user='steve' pwd='xyz' environment='prod'
role='*ALL' session=''>
</jdeRequest>
```

5.3.6 Implicit Transaction

An XML 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 request by specifying the name of a transaction set that has not previously been created. For an implicit start, the transaction set will be a manual commit set.

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

```
</jdeRequest>
```

5.3.7 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. Prepare, commit, and rollback requests to the database are made by using the endTransaction element. The transaction set is identified by the *trans* attribute. 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 indicated by the forward slash.

It is recommended that you 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' environment='prod'
role='*ALL' session=''
<endTransaction trans='t1' action='commit'/>
</jdeRequest>
```

Note: If startTransaction and endTransaction are in separate documents, one of these scenarios occurs:

The session attribute is not sent in the second document. In this case, the system uses the user ID, password, and environment to match the previous session.

The session number from the response of the first document is sent in the session attribute of the documents associated with the same transaction.

5.3.8 Terminate Session

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

```
<?xml version='1.0' ?>
<jdeRequest type='callmethod' user='steve' pwd='xyz' environment='prod'
role='*ALL' session=5665.931961929.454'>
<endSession/>
</jdeRequest>
```

5.4 XML Standards

In addition to ensuring that your XML documents have the required format elements (jdeRequest or jdeResponse Type, Establish Session, Expire Session, and Terminate Session), JD Edwards EnterpriseOne has standards for XML documents that are different from industry standards. Also, some special characters are reserved for XML and can't be used directly.

This section discusses:

- Decimal and comma separators.
- Data usage.
- Industry standards for special characters.

5.4.1 Decimal and Comma Separators

JD Edwards EnterpriseOne uses the decimal and thousands separators differently than XML industry standards. The decimal and thousands separators do not depend on use profile settings, jde.ini settings, or regional settings for the computer. When you write XML documents to interface with JD Edwards EnterpriseOne, you must always use the decimal character (.) (period) as a decimal separator, and a comma (,) as a thousands separator. The purpose of the separator standards is to achieve consistent interoperability policy and to prevent data corruption.

5.4.2 Date Usage

Different components of the XML foundation use different format codes and APIs to format these dates:

- to XML date
- from XML date
- to JDEDATE
- from JDEDATE

This table explains the formats that are used by each XML component supported by JD Edwards EnterpriseOne:

Component	Inbound Format	Inbound Outcome	Outbound Format	Outbound Outcome
XMLCallObject	F	YYYYMD	ESOSA	YYYY/MM/DD
XMLTransaction	F*	User Preference	ESOSA	YYYY/MM/DD
XMLList	B*	User Preference	NULL	User Preference

* Component ignores the format code

5.4.3 Industry Standards for Special Characters

In XML, some special characters are reserved for internal use, and to use these characters in data, you must replace them with entity or numeric references. This table shows the special characters that are reserved for XML along with the entity and numeric references that enable you to use a special character in your XML documents:

Character Name	Character	Entity Reference	Numeric Reference
Ampersand	&	&	&
Left angle bracket (less than)	<	<	<
Right angle bracket (greater than)	>	>	>
Straight quotation mark	"	"	"
Apostrophe	'	'	'
Percent	%	Not Applicable	%

Another way to use special characters in your XML documents is to use the CDATA section. Any text inside a CDATA section is ignored by the parser.

5.5 System Environment Configuration

Before you can use XML with JD Edwards EnterpriseOne, you must ensure that the ICU_DATA system environment variable is correctly defined on your JD Edwards EnterpriseOne system. If the ICU_DATA variable is not correctly defined, JD Edwards EnterpriseOne produces this error message:

The default Unicode converter could not be found within the jdenet_n.log on the OneWorld Enterprise Server.

For JD Edwards EnterpriseOne, the ICU conversion table, icu_data.dat, is generally located in system/locale/xml. Use the appropriate setting for your platform.

This section discusses:

- UNIX
- IBM i
- WIN32

5.5.1 UNIX

For UNIX systems, the ICU_DATA path is based on the ICU_DATA environment variable. The UNIX JD Edwards EnterpriseOne user login script sets the ICU_DATA environment variable to the directory location of the ICU resource file, incudata.dat. If the user login script does not set the ICU_DATA environment variable, you must define the ICU_DATA variable with a trailing slash, for example:

```
Export ICU_DATA=$SYSTEM/locale/xml/
```

Where $SYSTEM represents your JD Edwards EnterpriseOne install directory.

To support the loading of the JVM, verify the environment variable configuration for your platform.

5.5.1.1 HPUX

Verify these environment variable configurations for a HPUX platform:

```
export LD_LIBRARY_PATH=$SYSTEM/jre/1.4/lib/PA_RISC/server:
$SYSTEM/jre/1.4/lib/PA_
RISC:${LD_LIBRARY_PATH}
export SHLIB_PATH=$SYSTEM/jre/1.4/lib/PA_RISC/server:
$SYSTEM/jre/1.4/lib/PA_RISC:${SHLIB_PATH}
```

5.5.1.2 AIX

Verify these environment variable configurations for an AIX platform:

```
export LD_LIBRARY_PATH=$SYSTEM/jre/1.4/lib/bin:
$SYSTEM/jre/1.4/lib/bin/classic:${LD_LIBRARY_PATH}
export LIBPATH=$SYSTEM/jre/1.4/lib/bin:$SYSTEM/jre/1.4/lib/bin/classic:
${LIBPATH}
export SHLIB_PATH=$SYSTEM/jre/1.4/lib/bin:
$SYSTEM/jre/1.4/lib/bin/classic:${SHLIB_PATH}
```

5.5.1.3 SUN

Verify these environment variable configurations for a SUN platform:

```
export LD_LIBRARY_PATH=$SYSTEM/jre/1.4/lib/sparc/server:
$SYSTEM/jre/1.4/lib/sparc:${LD_LIBRARY_PATH}
export SHLIB_PATH=$SYSTEM/jre/1.4/lib/sparc/SERVER:
```

```
$SYSTEM/jre/1.4/lib/sparc:${SHLIB_PATH}
```

Note: Make sure that the server directory is first. The sparc directory has a libjvm.so just like the server directory, and the libjvm.so in the server directory is the directory you want to use.

5.5.1.4 LINUX

Verify these environment variable configurations for a LINUX platform:

```
export LD_LIBRARY_PATH=$SYSTEM/jre/1.4/lib/i386/server:
$SYSTEM/jre/1.4/lib/i386L:{LD_LIBRARY_PATH}
export SHLIB_PATH=$SYSTEM/jre/1.4/lib/i386/server:
$SYSTEM/jre/1.4/lib/i386:S{SHLIB_PATH}
```

5.5.2 IBM i

For IBM i systems, the ICU_DATA path is set when the ICU 1.6 conversion function is first called by the system. The system looks up Data Area BUILD_VER in the system library for the System Directory setting. For example:

System Directory: B9_S

The system appends locale/xml to the path specified in the BUILD_VER, and then uses this path as the ICU_DATA path. You must ensure the BUILD_VER data area is properly set to reflect the system directory setting.

5.5.3 WIN32

For WIN32 systems, the ICU_DATA path is set when the ICU 1.6 conversion function is first called. This logic is used on WIN32:

1. The system looks up the environment variable `JDE_B9_ICU_DATA`. If this environment is found, it becomes the path for the conversion files.
2. The system looks for this section in the jde.ini file:

   ```
   [XML]
   ICUPath=<<install>>/system/locale/xml
   ```
   If the ICUPath setting is found, it becomes the path for the conversion files.
3. If the system cannot find the ICUPath setting in the jde.ini file, the ICU_Path is:

   ```
   EXECUTABLE_DIRECTORY/./system/locale/xml
   ```
   The EXECUTABLE_DIRECTORY must be `<<install>>/system/bin32`.

Based on this logic, you usually do not need to set the JDE_B9_ICU_DATA ENVIRONMENT variable or the jde.ini file. You need to set the jde.ini ICUPath only when the location of the icudata.dat is different from system/locale/xml.

Note: The JD Edwards EnterpriseOne client install sets the environment variable JDE_B9_ICU_DATA.

5.6 XML Kernel Troubleshooting

If one or more XML kernels are not working properly, use these troubleshooting guidelines to ensure that your system is set up correctly:

- Check the kernel definition in the server jde.ini file.

Also check that the library name is correct for the platform on which you are running. Check the entry function name.

- Check that the kernel is allowed to start.

  Check the maxNumberOfProcesses and numberOfAutoStartProcesses values for the kernel in the server jde.ini file. It is not necessary to auto start kernels. To work with a particular kernel, the allowed number of processes should be one or more.

- If you have a large number of simultaneous requests that are made to a particular kernel type, increase the number of allowed processes for that kernel.

  This will not only reduce the turnaround time for requests but will also eliminate any Queue Full errors.

- If you are using XMLList kernel, check that the LREngine section is correctly set up in the server jde.ini file and that the specified path exists.

  Also, check that the JD Edwards EnterpriseOne user has write permission to this location.

- Check that the XML document is a well-formed XML document.

  To do this, use any XML editor or open the document in Microsoft Internet Explorer and check for errors.

- Check that the root of the input XML document is jdeRequest.

  All input XML documents should have jdeRequest as their root element.

- Check that valid user ID, password, and environment are provided in the XML document.

- Check that the request type in the XML document is correct. The allowed request types are callmethod, list, and trans for XMLCallObject, XMLList, and XMLTransaction kernels, respectively.

6 Understanding XML Dispatch

This chapter contains the following topics:

6.1 XML Dispatch

XML Dispatch is XML-based interoperability that runs as a JD Edwards EnterpriseOne kernel process. The XML Dispatch kernel is the central entry point for all XML documents. For incoming XML documents, XML Dispatch identifies the kind of document that comes into JD Edwards EnterpriseOne and sends the document to the appropriate kernel for processing. If XML Dispatch does not recognize the document, XML Dispatch sends the document to XTS to recognize and transform it into native JD Edwards EnterpriseOne format. After XTS transforms the document, the document is sent back to XML Dispatch to be sent to the appropriate kernel for processing. For outgoing documents, XML Dispatch remembers whether the request document was transformed into JD Edwards EnterpriseOne native format. If the incoming request was transformed, then the outgoing response document is sent to XTS for transformation from native JD Edwards EnterpriseOne format back into the format of the original request. After XTS transforms the document, the document is sent to XML Dispatch to distribute to the originator.

The XML Dispatch kernel is able to route and load balance the XML documents. For example, if you have many XML CallObject message types coming in at once, XML Dispatch tries to instantiate a new CallObject kernel. You set up the number of instances that a kernel can have in the jde.ini file. For example, if you set the number of instances for the CallObject kernel to five, if more than one CallObject document comes into JD Edwards EnterpriseOne, XML Dispatch sees that a particular kernel is busy and instantiates another one (up to five instances). XML Dispatch is able to recognize new kernel definitions (such as XAPI) if the kernel is defined in the jde.ini file. You are not required to change JDENET code when new kernels are added.

XML Dispatch is available on all platforms that are supported by JD Edwards EnterpriseOne.

6.2 XML Dispatch Processing

XML Dispatch receives standard JDENET messages (in the form of XML documents) from a transport driver or other jdenet_n. The communication between a transport and XML Dispatch is local inter-process communication (IPC) using JDENET APIs. The communication between XML Dispatch and XTS and between XML Dispatch and XML kernels can be either IPC or remote network using JDENET APIs.

XML Dispatch parses the XML document and sends the document to the appropriate JD Edwards EnterpriseOne kernel for processing.

6.3 XML Dispatch Recognizers

XML Dispatch uses recognizers to determine how to handle incoming and outgoing XML documents. If XML Dispatch recognizes an incoming XML document as being in JD Edwards EnterpriseOne native XML format, the XML document is parsed and sent to the appropriate kernel. For outgoing documents, the recognizer determines whether an XML document can be left as JD Edwards EnterpriseOne native XML format or whether it must be transformed.

You can add more than one recognizer to XML Dispatch to recognize different XML grammar. XML Dispatch recognizes the these types:

- jdeRequest
- jdeResponse
- jdeWorkflow

The XML Dispatch recognizer raises DocIsRecognized exception on document identification to stop further parsing.

You can write a recognizer that is able to recognize other types of XML documents. The specification for the type is configured in the jde.ini file.

6.4 XML Dispatch Transports

As part of XML Dispatch, you can write a transport. Transports communicate with external systems using mechanisms such as MQ WebSphere, MSMQ, HTTP, TCP/IP, and so on. Transport processes must run on the same machine as XML Dispatch. To develop a custom transport to communicate with JD Edwards EnterpriseOne, use these APIs:

- jdeTransportInit
- jdeTransportMessagePut
- jdeTransportMessageGet
- jdeTransportDoIExit

The transport APIs assume a polling model, which means calls to put or receive messages are given without a time-out.

6.5 XML Dispatch jde.ini File Configuration

The XML Dispatch kernel must be defined in the jde.ini file.

6.5.1 [JDENET_KERNEL_DEF22]

These settings are for a Microsoft Windows platform:

```
krnlName=XML DISPATCH KERNEL
dispatchDLLName=xmldispatch.dll
dispatchDLLFunction=_XMLDispatch@28
maxNumberOfProcesses=1
numberOfAutoStartProcesses=1
```

This table provides the different .dll extensions for other platforms.

Platform	dispatchDLLName	dispatchDLLFunction
IBM i	XMLDSPATCH	?XMLDispatch?
HP9000	libxmldispatch.sl	?XMLDispatch?
SUN or RS6000	libxmldispatch.so	?XMLDispatch?

XML Dispatch uses the settings in the [XMLLookupInfo] section of the jde.ini file to route XML documents to the corresponding XML kernels. The system uses three keywords (XMLRequestN, XMLKernelMessageRangeN, and XMLKernelHostN) to map a pair that consists of an XML request and an XML kernel. A description of the settings in the [XMLLookupInfo] section are explained in this table:

Setting	Purpose
XMLRequestTypeN=	Identifies the type of message to be processed.
XMLKernelMessageRangeN=	A hard-coded number that identifies the kernel message range.
XMLKernelHostNameN=	The name of the host.
XMLKernelPortN=	Value is 0 or 1. To indicate a local host, enter 0. To indicate a remote host, enter 1.
XMLKernelRplyN=	Value is 0 or 1, with 1 as the default value. A value of 0 indicates no reply is required. A value of 1 indicates a reply should be returned to the originator. **Note:** XMLKernelRplyN setting is not required for list, callmethod, and trans. The reply setting is an implied 1. XMLService does not send a response, and the setting for XMLKernelReplyN should be zero (0). Where N starts with 1, and multiple groups of these keys can be in this section.

6.5.2 [XMLLookupInfo]

The [XMLLookupInfo] section should have six groupings, as illustrated in this example:

```
[XMLLookupInfo]
XMLRequestType1=list
XMLKernelMessageRange=5257
XMLKernelHostName1=local
XMLKernelPort1=0


XMLRequestType2=callmethod
XMLKernelMessageRange2=920
XMLKernelHostName2=local
```

```
XMLKernelPort2=0

XMLRequestType3=trans
XMLKernelMessageRange3=5001
XMLKernelHostName3=local
XMLKernelPort3=0

XMLRequestType4=JDEMSGWFINTEROP
XMLKernelMessageRange4=4003
XMLKernelHostName4=local
XMLKernelPort4=0
XMLKernelReply4=0

XMLRequestType5=xapicallmethod
XMLKernelMessageRange5=14251
XMLKernelHostName5=local
XMLKernelPort5=0
XMLKernelReply5=0

XMLRequestType6=realTimeEvent
XMLKernelMessageRange6=14251
XMLKernelHostName6=local
XMLKernelPort6=0
XMLKernelReply6=0

XMLRequestType7=ube
XMLKernelMessageRange7=380
XMLKernelHostName7=local
XMLKernelPort7=0
XMLKernelReply7=1
```

The XML Dispatch kernel uses these two additional settings:

```
[XML DISPATCH]
PollIntervalMillis=3000

[XTS]
ResponseTimeout=600
```

The PollIntervalMillis setting is the number of milliseconds that the XML Dispatch kernel sleeps during inactivity when it is waiting on responses from other XML kernels such as XML CallObject. The lower this value, the more CPU cycles the XML Dispatch kernel uses when waiting for responses.

The ResponseTimeout setting is the number of seconds that the XML Dispatch kernel waits for a response from other XML kernels, such as CallObject) before giving up on the response.

6.6 XML Dispatch Error Handling

XML Dispatch handles three types of errors. This table identifies the errors and how XML Dispatch handles the error:

XML Dispatch Error	How XML Dispatch Handles the Error
An error occurs while XML dispatch, XTS, and the XL kernel processes are exchanging data. For example, communication is broken.	XML Dispatch generates an error report, which is an XML document that describes the error.
An error occurs while the parser or XTS is processing an XML document. For example, a syntax error, an invalid request, and so on.	XML Dispatch generates an error report that is based on the error message that is generated by either the parser or XTS.
An error occurs while an XML kernel is processing an XML document. For example, the user name is invalid, the transaction is rolled back, and so on.	XML Dispatch uses XTS to transform the XML kernel generated error report when necessary.

XML Dispatch sends generated error reports to the corresponding transport process.

6.7 Submit a UBE from XML

You can use the XML interoperability solution (XML Callobject and XML List) to submit a UBE that requests inbound XML. The COM connector, Dynamic Java connector, and Java connector support inbound synchronous XML requests. You can use the run RUNUBEMXL command; however, this command works only on the JD Edwards EnterpriseOne Enterprise server.

6.7.1 Prerequisites

Before you request an inbound XML, do the following:

- Configure the JD Edwards EnterpriseOne server jde.ini file, [XMLLookupInfo] section for XML Request type 7, as illustrated here:

  ```
  [XMLLookupInfo]
  XMLRequestType7=ube
  XMLKernelMessageRange7=380
  XMLKernelHostName7=local
  XMLKernelPort7=0
  XMLKernelReply7=1
  ```

- Create a processing option that contains data selection and data sequencing that you want and submit from batch version to make sure that you obtain the desired result.

  For example, R0010P creates a new version, ABCD (where company=00001.)

See Also

- "Requesting Inbound XML Using COM Server" in the *JD Edwards EnterpriseOne Tools Connectors Guide*.
- "Inbound XML Request Using the Dynamic Java Connector" in the *JD Edwards EnterpriseOne Tools Connectors Guide*.
- "Inbound XML Request Using the Java Connector" in the *JD Edwards EnterpriseOne Tools Connectors Guide*.

7

Understanding XML Transformation Service

This chapter contains the following topics:

- Section 7.1, "XML Transformation Service"
- Section 7.2, "XTS Process"
- Section 7.3, "Custom Selectors"
- Section 7.4, "XTS jde.ini File Configuration"

7.1 XML Transformation Service

The JD Edwards EnterpriseOne XML transformation system (XTS) uses extensible stylesheet language (XSL) to transform XML documents to the format that is required by JD Edwards EnterpriseOne. XTS also transforms JD Edwards EnterpriseOne response XML documents back to the XML format of the original request.

XTS is a multi-threaded Java process that runs as a JD Edwards EnterpriseOne kernel process. Upon system startup, the XTS kernel library loads a Java virtual machine (JVM). Once the JVM is loaded, the server proxy is started. Java Runtime Environment (JRE) v. 1.4 is included with the JD Edwards EnterpriseOne software.

XTS is available on all platforms that JD Edwards EnterpriseOne supports.

7.2 XTS Process

When the JD Edwards EnterpriseOne XML Dispatch kernel receives an XML document that it does not recognize, it sends the document to XTS for transformation. XTS reads the XSL, transforms the document to a format that is compatible with JD Edwards EnterpriseOne, and sends the document back to the XML Dispatch kernel for processing. When the JD Edwards EnterpriseOne response comes into XML Dispatch, XML Dispatch remembers that the document needs to be transformed from the JD Edwards EnterpriseOne XML format and sends the document to XTS for transformation. XTS transforms the JD Edwards EnterpriseOne XML document back to your original XML format and sends the document to XML Dispatch for distribution to you.

Native XML format is the XML format that is defined by JD Edwards EnterpriseOne and is documented in this guide. All XML documents coming into JD Edwards EnterpriseOne must be in native XML format. The JD Edwards EnterpriseOne kernel processes (such as, XML CallObject, XML trans, XML list, and so on) can only process XML documents that are in native format. As part of the XTS solution, JD Edwards EnterpriseOne provides a selector that determines whether a non-JD Edwards EnterpriseOne XML document can be transformed. A selector is code that looks at an

XML document to see if it recognizes the document. If the selector recognizes the XML document, the selector is able to associate the XML document with a stylesheet that is provided for transformation. The selector is able to transform Version 1 XML format into JD Edwards EnterpriseOne native XML format. Version 1 XML format is XML format that is defined by JD Edwards EnterpriseOne but has been modified to be tool friendly. Native XML format uses a field name that is preceded by parameter name. Version 1 XML format uses just the field name.

7.2.1 Example: JD Edwards EnterpriseOne Native XML Format

This sample code shows JD Edwards EnterpriseOne native XML format:

```
<xml version='1.0'?>
<jdeRequest pwd='mike' type='callmethod' user='mike' environment='DV810'>
<callMethod app='test' name='GetPhone'>
<params>
<param name='mnAddressnumber'>4242a</param>
<param name= 'mnLinenumberid'></param>
<param name= 'cIncludeexcludecode2'></param>
<param name= 'szPhonenumber'>/param>
<param name= 'szPhoneareacode1'></param>
<param name= 'mnOKtoDelete'></param>
<param name= 'szPhonenumberType'></param>
</params>
</callMethod>
</jdeRequest>
```

7.2.2 Example: JD Edwards EnterpriseOne Version 1 XML Format

This sample code shows Version 1 XML format:

```
<?xml version=1.0 ?>
<intBPAPI>
<dsControl>
 <dsLogin>
  <User>JDESVR</User>
  <Password>JDESVR</Password>
  <Environment>ADEVNIS2</Environment>
  <Session />
 </dsLogin>
 <dsAPI>
  <Noun>jdeSalesOrder</Noun>
  <Verb>Create</Verb>
  <Version>1.1</Version>
 </dsAPI>
 <dsTranslation>
  <InMap />
  <OutMap />
 </dsTranslation>
</dsControl>
<dsData>
 <callMethod_GetLocalComputerId app="NetComm" runOnError="no">
  <szMachineKey id="" />
  <onError_GetLocalComputerId abort="yes" />
 </callMethod_GetLocalComputerId>
<callMethod_F4211FSBeginDoc app="NetComm" runOnError="no">
 <mnCMJobNumber id="" />
 <cCMDocAction>A</cCMDocAction>
 <cCMProcessEdits>1</cCMProcessEdits>
 <szCMComputerID idref="2" />
```

```
 <cCMUpdateWriteToWF>2</cCMUpdateWriteToWF>
 <szCMProgramID>NetComm</szCMProgramID>
 <szCMVersion>NetComm</szCMVersion>
 <szOrderType>SQ</szOrderType>
 <szBusinessUnit>M30</szBusinessUnit>
 <mnAddressNumber>4242</mnAddressNumber>
 <szReference>2</szReference>
 <cApplyFreightYN>Y</cApplyFreightYN>
 <szCurrencyCode>CAD</szCurrencyCode>
 <cWKSourceOfData />
 <cWKProcMode>1</cWKProcMode>
 <mnWKSuppressProcess>0</mnWKSuppressProcess>
 <onError_F4211FSBeginDoc abort="yes">
  <callMethod_F4211ClearWorkFile app="NetComm" runOnError="yes">
   <mnJobNo idref="1" />
   <szComputerID idref="2" />
   <mnFromLineNo>0</mnFromLineNo>
   <mnThruLineNo>0</mnThruLineNo>
   <cClearHeaderWF>2</cClearHeaderWF>
   <cClearDetailWF>2</cClearDetailWF>
   <szProgramID>NetComm</szProgramID>
   <szCMVersion>ZJDE0001</szCMVersion>
  </callMethod_F4211ClearWorkFile>
 </onError_F4211FSBeginDoc>
</callMethod_F4211FSBeginDoc>
 <callMethod_F4211FSEditLine app="NetComm" runOnError="yes">
  <mnCMJobNo idref="1" />
  <cCMLineAction>A</cCMLineAction>
  <cCMProcessEdits>1</cCMProcessEdits>
  <cCMWriteToWFFlag>2</cCMWriteToWFFlag>
  <szCMComputerID idref="2" />
  <mnLineNo>1</mnLineNo>
  <szItemNo>1001</szItemNo>
  <mnQtyOrdered>5</mnQtyOrdered>
  <cSalesTaxableYN>N</cSalesTaxableYN>
  <szTransactionUOM>EA</szTransactionUOM>
  <szCMProgramID>1</szCMProgramID>
  <szCMVersion>ZJDE0001</szCMVersion>
  <cWKSourceOfData />
  <onError_F4211FSEditLine abort="no" />
 </callMethod_F4211FSEditLine>
 <callMethod_F4211FSEndDoc app="NetComm" runOnError="no">
  <mnCMJobNo idref="1" />
  <szCMComputerID idref="2" />
  <szCMProgramID>NetComm</szCMProgramID>
  <szCMVersion>ZJDE0001</szCMVersion>
  <cCMUseWorkFiles>2</cCMUseWorkFiles>
  <mnSalesOrderNo id="" />
  <szKeyCompany id="" />
  <mnOrderTotal id="" />
  <onError_F4211FSEndDoc abort="no">
   <callMethod_F4211ClearWorkFile app="NetComm" runOnError="yes">
    <mnJobNo idref="1" />
    <szComputerID idref="2" />
    <mnFromLineNo>0</mnFromLineNo>
    <mnThruLineNo>0</mnThruLineNo>
    <cClearHeaderWF>2</cClearHeaderWF>
    <cClearDetailWF>2</cClearDetailWF>
    <szProgramID>NetComm</szProgramID>
    <szCMVersion>ZJDE0001</szCMVersion>
```

```
   </callMethod_F4211ClearWorkFile>
  </onError_F4211FSEndDoc>
 </callMethod_F4211FSEndDoc>
 <returnParams failureDestination="error" successDestination="success"
runOnError="yes">
  <mnOrderNo idref="3" />
  <szOrderCo idref="4" />
  <mnWKOrderTotal idref="5" />
 </returnParams>
 <onError abort="yes">
  <callMethod_F4211ClearWorkFile app="NetComm" runOnError="yes">
   <mnJobNo idref="1" />
   <szComputerID idref="2" />
   <mnFromLineNo>0</mnFromLineNo>
   <mnThruLineNo>0</mnThruLineNo>
   <cClearHeaderWF>2</cClearHeaderWF>
   <cClearDetailWF>2</cClearDetailWF>
   <szProgramID>NetComm</szProgramID>
   <szCMVersion>ZJDE0001</szCMVersion>
  </callMethod_F4211ClearWorkFile>
 </onError>
</dsData>
</intBPAPI>
```

7.3 Custom Selectors

You can build a selector to transform your XML format into JD Edwards EnterpriseOne native XML format. If you write a custom selector, include both request and response extensible stylesheet language transformation (XSLT) documents.

Inside the Java file, the system uses two APIs to select templates. Use the boolean fetchTemplates API to fetch the appropriate XSLT document for the request document. Public boolean fetchTemplates throws IXTSMTemplateSelector.TemplateFetchException, XTSXMLParseException. This sample shows how to use this API:

```
fetchTemplates(XTSDocument  inXML,  IXTSMSelectionInfo  info)
```

Use the Public void fetchTemplates to fetch the appropriate XSLT document for the response document. Public void fetchTemplates throws IXTSMTemplateSelector.TemplateFetchException.

```
fetchTemplates(IXTSMSelectionInfo  info)
```

Note: Ensure that your custom selector is accessible in the ClassPath.

7.3.1 XTS APIs

When you write a custom selector, you can use these APIs to interface with JD Edwards EnterpriseOne:

- IXTSMTemplateSelector
- IXTSMTemplateSelector.TemplateFetchException

7.3.2 Example: Creating a Selector

This code was written by JD Edwards EnterpriseOne to build the Version 1 XML selector. This sample code uses the XTS.jar file. You can use this code as a sample for creating your selector:

```
File: XTSMJDETemplateSelector.java
//
////////////////////////////////////////////////////////////////////////////////
package com.jdedwards.xts.xtsm;
import com.jdedwards.xts.xtsr.IXTSRepository;
import com.jdedwards.xts.xtsr.IXTSRKey;
import com.jdedwards.xts.xtsr.XTSRException;
import com.jdedwards.xts.xtsr.XTSRInvalidKeyStringException;
import com.jdedwards.xts.xtsr.XTSRInvalidKeyFieldException;
import com.jdedwards.xts.xtsr.XTSRKeyNotFoundException;
import com.jdedwards.xts.XTSDocument;
import com.jdedwards.xts.XTSFactory;
import com.jdedwards.xts.XTSLog;
import com.jdedwards.xts.XTSConfigurationException;
import com.jdedwards.xts.XTSXMLParseException;
import com.jdedwards.xts.xtsm.IXTSMTemplateSelector;
import com.jdedwards.xts.xtse.IXTSEngine;
import com.jdedwards.xts.xtse.IXTSECompiledProcessor;
import java.util.List;
import org.w3c.dom.*;

/**
* This class is the  Template Selector. It recognizes
* JD Edwards EnterpriseOne standard XML documents and returns the
* appropriate XSL stylesheets necessary for transformation.
*/
public class XTSMJDETemplateSelector implements IXTSMTemplateSelector
{
 /** Class constructor. */
 public XTSMJDETemplateSelector()
 {
  XTSLog.trace(XTSMJDETemplateSelector()'', 3);
  // get repository reference
  XTSFactory factory = XTSFactory.getInstance();
  m_repository = factory.createXTSRepository();
 }

 /**
 * Fetch the appropriate XSLT documents and IXTSECompiledProcessors as
 * indicated by the TPT stored in the <code>info</code> parameter.
 * @param info - Selection Info that contains TPI and should be modified
 * by the selector to specify transformation information.
 * @exception IXTSMTemplateSelector.TemplateFetchException - thrown
 * if an error occurs when extracting information from the
 * inclement.
 */
 public void fetchTemplates(IXTSMSelectionInfo info)
    throws IXTSMTemplateSelector.TemplateFetchException
 {
  XTSLog.trace("XTSMJDETemplateSelector.fetchTemplates(XTSMSelectionResult)",
3);
  NodeList nodes = info.getTPIElement().getElementsByTagName(JDE_TS_XTSR_KEY);
  int numNodes = nodes.getLength();
  for(int i = 0; i < numNodes; i++)
  {
```

```
    // extract key info & create a key
    IXTSRKey key = createKeyFromNode((Element)nodes.item(i));

    // fetch the doc and add it to the list
    try
    {
     info.getXSLList().add(m_repository.fetch(key));
    }
    catch (XTSRKeyNotFoundException e)
    {
     throw new IXTSMTemplateSelector.TemplateFetchException(
        "Selected XTSRKey not found in repository: "
        + JDE_TS_XTSR_KEY);
    }
    catch (XTSRException e)
    {
     throw new IXTSMTemplateSelector.TemplateFetchException(
        "Unable to fetch the XSL document specified within '"
        + JDE_TS_XTSR_KEY +
        "' from the XTSRepository");
    }
 }
}

/**
* Fetch the appropriate XSLT documents and compiled processors for
* the given document.
* @param inXML - the XTSDocument to try to recognize.
* @param info - Selection Info object to be modified by selector to
* indicate transformation information.
* @return - <code>true</code> if the selector has recognized the
* document and specified the appropriate selection info using
* <code>info</code>, <code>false</code> otherwise.
* @exception TemplateFetchException - thrown when an error occurs
* when trying to recognize the DOM.
* @exception XTSXMLParseException - thrown if <inXML> could not be
* parsed.
*/
public boolean fetchTemplates(XTSDocument inXML,
               IXTSMSelectionInfo info)
 throws IXTSMTemplateSelector.TemplateFetchException,
     XTSXMLParseException
{
 XTSLog.trace("XTSMJDETemplateSelector.fetchTemplates(Document, Element)", 3);
 boolean recognized = false;
 Document inDOM = inXML.getDOM();
 // see if an XTSR key is specified within the document:
 NodeList nodeList = inDOM.getElementsByTagName(JDE_XTSR_KEY);
 if (nodeList.getLength() > 0)
 {
   try
   {

     // extract key info & create a key
     IXTSRKey key = createKeyFromNode((Element)nodeList.item(0));

     // add transformation path information to outElement
     createNodeChildFromKey(info.getTPIElement(), key);

     // fetch the doc and add it to the list
```

```
    info.getXSLList().add(m_repository.fetch(key));

    info.setResultXML(true);
    info.setPathInfoStored(false);
    recognized = true;
   }
   catch (XTSRException e)
   {
    throw new IXTSMTemplateSelector.TemplateFetchException(
         "Unable to fetch the XSL document specified within '"
         + JDE_XTSR_KEY +
         "' from the XTSRepository");
   }
   catch (XTSRKeyNotFoundException e)
   {
    throw new IXTSMTemplateSelector.TemplateFetchException(
         "Key specified in TPI not found in repository"
         + JDE_XTSR_KEY);
   }
 }
 else // no XTSR key, so look for JDE information:
 {
   nodeList = inDOM.getElementsByTagName(JDE_INT_BPAPI);
   if (nodeList.getLength() != 0)
   {
    // add transformation path information to outElement
    createNodeChildFromKey(info.getTPIElement(), getVersion1toNativeKey());

    // fetch the doc and add it to the list
    info.getXSLList().add(getVersion1toNativeXSL());

    info.setResultXML(true);
    info.setPathInfoStored(true);
    recognized = true;
   }
 }
 return recognized;
}
/**
* Extracts XTSRKey information from the given node, and creates an
* instance of IXTSRKey based on that information.
* @return - the new IXTSRKey.
* @param element - Element that contains the key information.
* @exception XTSMUnrecognizedElementException - thrown if the
* Element format is unrecognized.
*/
protected IXTSRKey createKeyFromNode(Element element)
     throws XTSMUnrecognizedElementException
{
 XTSLog.trace("XTSMJDETemplateSelector.createKeyFromNode(Element)", 4);
 IXTSRKey key = null;
 boolean request = false;
 boolean response = false;
 if (element.getNodeName().equals(JDE_XTSR_KEY))
 {
   request = true;
 }
 else if (element.getNodeName().equals(JDE_TS_XTSR_KEY))
 {
   response = true;
```

```
  }
  if (request || response)
  {
    key = m_repository.createKey();
    try
    {
     String keyString = element.getAttribute(JDE_XTSR_KEY_ATTRIBUTE);
     key.setFieldsFromString(keyString);
     if (key.getFieldValue(SUBTYPE_FIELD).length() == 0)
     {
       if (request)
       {
        key.setFieldValue(SUBTYPE_FIELD, SUBTYPE_REQUEST);
       }
       else
       {
        key.setFieldValue(SUBTYPE_FIELD, SUBTYPE_RESPONSE);
       }
     }
    }
    catch (XTSRInvalidKeyStringException e)
    {
     throw new XTSMUnrecognizedElementException(
          "Specified '" + JDE_XTSR_KEY +
          "' element format is invalid for this XTSRepository");
    }
    catch (XTSRInvalidKeyFieldException e)
    {
     throw new XTSConfigurationException(
          "Specified '" + SUBTYPE_FIELD +
          "' field name not supported by repository key");
    }
  }
  return key;
 }
 /**
 * Creates a node that contains the key fields values and appends it
 * to the given parentNode.
 * @param parentNode - Node to which the key information should be
 * appended.
 * @param key - Key information to store in the node.*/

 protected void createNodeChildFromKey(Node parentNode, IXTSRKey key)
 {
  XTSLog.trace("XTSMJDETemplateSelector.createKeyFromNode(Node,IXTSRKey)", 4);
  try
  {
    IXTSRKey keyClone = key.getRepository().createKey();
    keyClone.setFieldsFromString(key.getFieldsString());

    // Do not store the sub type, clear it here:
    keyClone.setFieldValue(SUBTYPE_FIELD, "");

    // create new node and append it to the provided element:
    Element element = (Element)parentNode.getOwnerDocument().createElement
(JDE_TS_XTSR_KEY);
    element.setAttribute(JDE_XTSR_KEY_ATTRIBUTE, keyClone.getFieldsString());
    parentNode.appendChild(element);
  }
  catch (XTSRInvalidKeyStringException e)
```

```
 {
   XTSLog.log("Unexpected ");
   XTSLog.log(e);
   throw new RuntimeException("Unexpected Exception: " + e.toString());
 }
}

/**
* Returns the key of the stylesheet to use in converting
* JD Edwards EnterpriseOne version 1 documents into EnterpriseOne native
* documents.
* @return - The key for the XSL stylesheet.
*/
protected IXTSRKey getVersion1toNativeKey()
{
 XTSLog.trace("XTSMJDETemplateSelector.getVersion1toNativeKey()", 5);
 if (null == m_version1ToNativeKey)
 {
   try
   {
    // create standard xsl XTSRKey:
    m_version1ToNativeKey = m_repository.createKey();
    m_version1ToNativeKey.setFieldsFromString(V1_TO_NATIVE_KEY);
   }
   catch (XTSRInvalidKeyStringException e)
   {
    String error = "XTSRKey necessary for JDE template selection is invalid: "
          + V1_TO_NATIVE_KEY;
    XTSLog.log(error);
    XTSLog.log(e);
    throw new XTSConfigurationException(error);
   }
 }
 return m_version1ToNativeKey;
}

/**
* Returns the XTSDocument which contains the XSL stylesheet for
* converting JD Edwards EnterpriseOne version 1 documents into JD Edwards
* EnterpriseOne native documents.
* @return - XTSDocument containing the XSL stylesheet.
*/
protected IXTSECompiledProcessor getVersion1toNativeXSL()
{
 XTSLog.trace("XTSMJDETemplateSelector.getVersion1toNativeXSL()", 5);
 if (null == m_version1ToNativeXSL)
 {
   XTSDocument xsl = null;
   Try
   {
    xsl = m_repository.fetch(getVersion1toNativeKey());
    IXTSEngine engine = XTSFactory.getInstance().createXTSEngine();
    m_version1ToNativeXSL = engine.createCompiledProcessor(xsl);
   }
   catch (XTSRException e)
   {
    String error = "Unable to fetch selected template from the repository:";
    XTSLog.log(error);
    XTSLog.log(e);
    throw new XTSConfigurationException(error + e.toString());
```

```
    }
    catch (XTSRKeyNotFoundException knfe)
    {
     String error = "Selected template XTSRKey not found in repository:";
     XTSLog.log(error);
     XTSLog.log(knfe);
     throw new XTSConfigurationException(error + knfe.toString());
    }
    catch (XTSXMLParseException pe)
    {
     String error = "Invalid XSL document in repository";
     XTSLog.log(error);
     XTSLog.log(pe);
     throw new XTSConfigurationException(error + pe.toString());
    }
  }
  return m_version1ToNativeXSL;
}

/** Reference to the XTSRepository */
private IXTSRepository m_repository = null;

/** Key for converting version 1 documents to native documents. */
private IXTSRKey m_version1ToNativeKey = null;

/** Compiled XSL Stylesheet for converting version 1 docs to
 *  native docs. */
private IXTSECompiledProcessor m_version1ToNativeXSL = null;

/** Field Value for the XTSRKey that indicates the document is an XSL doc */
private static final String DOC_TYPE_XSL = "XSL";

/** Element name that indicates the DOM is a Version 1 document */
private static final String JDE_INT_BPAPI = "intBPAPI";

/** Element name that indicates the DOM is a request and not a
 *  response or error. */
private static final String JDE_REQUEST = "jdeRequest";

/** Element name that indicates the DOM is a response */
private static final String RESPONSE = "jdeResponse";

/** Element name that specifies an XTSRKey to use in transforming
 *  the document. */
private static final String JDE_XTSR_KEY = "jdeXTSRKey";

/** The attribute of the <code>JDE_XTSR_KEY</code> element that
 *  stores the XTSRKey string value */
private static final String JDE_XTSR_KEY_ATTRIBUTE = "key";

/** XTSRKey field name that specifies the sub-type of the XML
 * document. Normal values for the sub-type are defined by
 * <code>SUBTYPE_REQUEST</code> and <code>SUBTYPE_RESPONSE</code> */
private static final String SUBTYPE_FIELD = "SUB_TYPE";

/** XTSRKey field name which specifies the type of the XML document.
 *  The normal value is defined by <code>DOC_TYPE_XSL</code> */
private static final String FIELD_TYPE = "TYPE";

/** XTSRKey field name which specifies the format (or owner) of the
```

```
 *  XML document. The normal value recognized by this selector is
 * 'JDE' */
private static final String FIELD_FORMAT = "FORMAT";

/** XTSRKey field name that specifies the particular transformation
 *  that the XSL document will perform. This selector uses
 *  'V1_NATIVE' for transformations between JD Edwards EnterpriseOne Version 1
 *  XML documents and JD Edwards EnterpriseOne native version documents. */
private static final String FIELD_ID = "ID";

/** The string representation of the XTSRKey for the XSL document to
 *  format JD Edwards EnterpriseOne version 1 request documents into
 *  JD Edwards EnterpriseOne native request documents. */
private static final String V1_TO_NATIVE_KEY = "XSL-JDE-V1_NATIVE-REQUEST";

/** XTSRKey field <code>SUBTYPE_FIELD</code> value that indicates
 *  the XSL document will transform jdeRequest documents. */
private static final String SUBTYPE_REQUEST = "REQUEST";

/** XTSRKey field <code>SUBTYPE_FIELD</code> value that indicates
 *  the XSL document will transform jdeResponse documents. */
private static final String SUBTYPE_RESPONSE = "RESPONSE";

/** Element name stored within the Transformation Path Information
 * (TPI)that specifies the XTSRKey used to transform the document. */
private static final String JDE_TS_XTSR_KEY = "XTSJDETemplateKey";

private static class XTSMUnrecognizedElementException
    extends IXTSMTemplateSelector.TemplateFetchException
{
 public XTSMUnrecognizedElementException(String text)
 {
   super(text);
 }
}
}
```

7.4 XTS jde.ini File Configuration

The XTS Kernel must be defined in the server jde.ini file. The name of the configuration file is retrieved from the config_file system variable in the JVM. These property settings are part of a configuration file other than jde.ini. The jde.ini file does not require any special configurations other than to define the XTS Kernel.

7.4.1 [JDENET_KERNEL_DEF23]

These setting are for a Microsoft Windows platform:

```
krnlName=JDEXTS KERNEL
dispatchDLLName=xtskrnl.dll
dispatchDLLFunction=_JDEK_DispatchXTSMessage@28
maxNumberOfProcesses=1
numberOfAutoStartProcesses=0
```

This table provides the different .dll extensions for other platforms:

Table Column Heading	Dispatch DLL Name	Dispatch DLL Function
IBM i	XTSKRNL	JDEK_DispatchXTS

Table Column Heading	Dispatch DLL Name	Dispatch DLL Function
HP9000B	libxtskrnl.sl	JDEK_DispatchXTS
SUN or RS6000	libxtskrnl.so	JDEK_DispatchXTS

Other jde.ini File settings include:

- [JDENET]
- [XTSRepository]
- [XTS]

7.4.2 [JDENET]

Configure this setting:

```
maxKernelRanges=24
```

Note: For the XTS kernel to run, set the maxKernelRanges setting to 23 or higher.

7.4.3 [XTSRepository]

Configure these settings:

```
XSL-JDE-V1_NATIVE-REQUEST=ml.xsl
XSL-JDE-V1_NATIVE-RESPONSE=lm.xsl
```

Note: The first setting is the JD Edwards EnterpriseOne default value that enables XSL to transform the request document from Version 1 to native. The second setting is the JD Edwards EnterpriseOne default value that enables XSL to transform the response document from native to version 1.

You can provide your XSL files either at this location or any other location as long as your selector can find and access your XSL. To add your XSL files to this location, use these naming conventions, where Filename is the name of your XSL documents:

XSL-JDE-Filename-REQUEST=

XSL-JDE-Filename-RESPONSE=

7.4.4 [XTS]

This is an example setting:

```
XTSTemplateSelector1=com.jdedwards.xts.xtsm.XTSMJDETemplateSelector
XTSTraceLevel=2
```

Note: The XTSTemplateSelector1 setting is the JD Edwards EnterpriseOne default template selector for providing XSL to transform between Version 1 and native format.

You can add your custom template selector to this section. For example, your template selector setting could be defined as follows:

XTSTemplateSelector2=com.customer.CustomTemplateSelector

The XTSTraceLevel=2 setting defines the level of XTS logging.

---

8 Understanding XML CallObject

This chapter contains the following topics:

8.1 XML CallObject

XML CallObject is XML-based interoperability that runs as a JD Edwards EnterpriseOne kernel process. You can also use XML CallObject with a messaging adapter. Some features of XML CallObject include:

- The ability to make business function calls to JD Edwards EnterpriseOne using XML documents.
- Business function templates and the ability to create your own templates.
- The ability to call multiple business functions using a single XML document.
- A simpler way of interfacing with JD Edwards EnterpriseOne as compared to using COM or Java APIs.

8.2 XML CallObject Templates

XML CallObject provides a blank template that you can complete to make CallObject requests for a given business function. You also have the option of creating your own custom XML documents.

To request an XML template for a given business function, you create an XML document that is a callMethod request type. When you make a CallObject template request, the response is the template that has information about all of the function parameters but is not populated with data values. The user, password, and session attribute values are blank so that you can cache the response for later use.

A CallObject template request is an exception to the convention that a jdeRequest returns a jdeResponse. Instead of data, you receive the template, which you use to make another callMethod request. When you request a CallObject template, the request for the template is the only request that can be made in the XML document. The XML document must include the business function.

This example illustrates a request for a CallObject template:

```
<?xml version='1.0' ?>
<jdeRequest type='callmethod' user='steve' pwd='xyz' environment='prod'
role="*ALL" session=''>
<callMethodTemplate name='myfunc' app='P42101'/>
</jdeRequest>
```

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

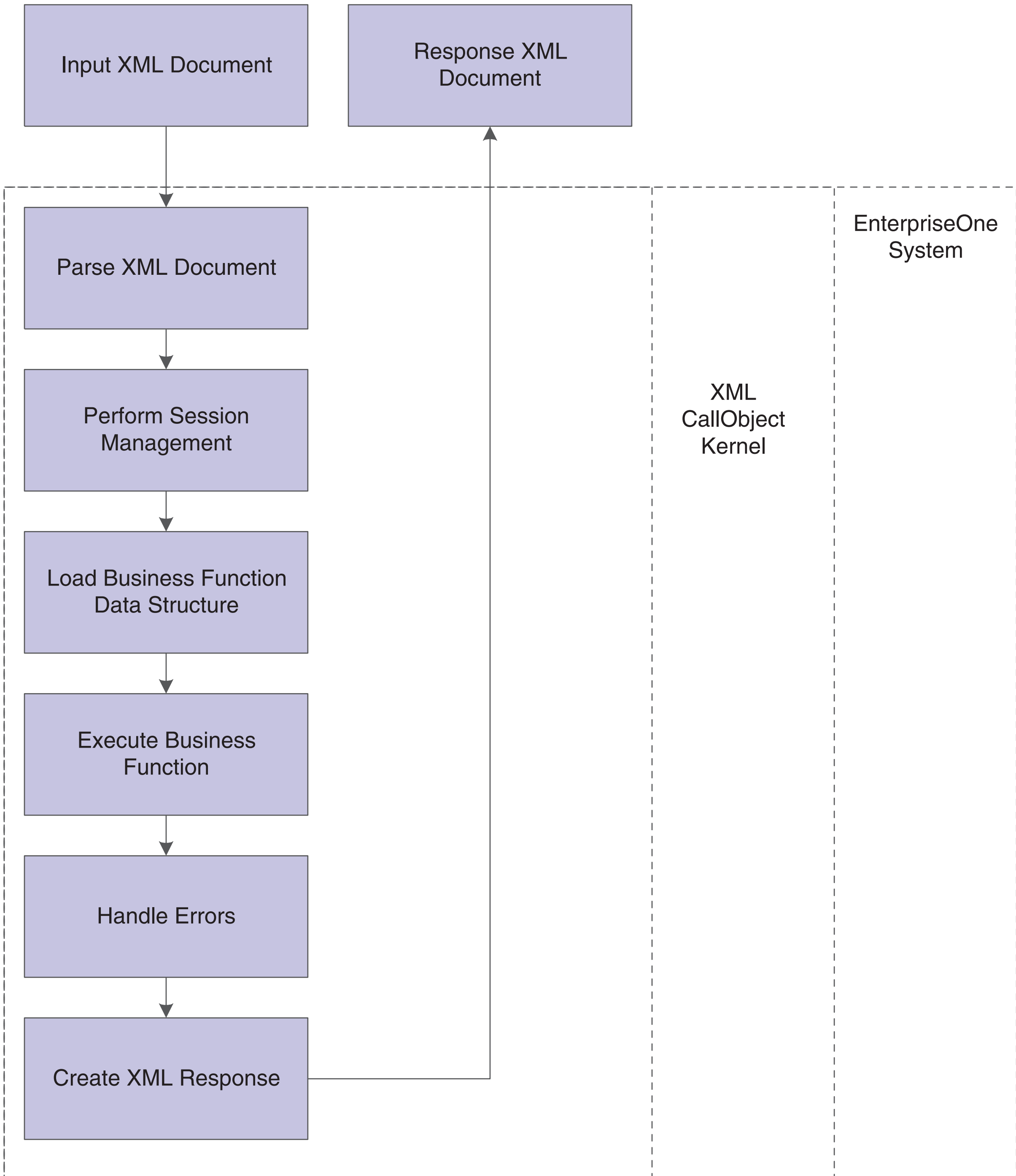
```
<?xml version='1.0' ?>
<jdeRequest type='callmethod' user='' pwd='' environment='prod' role='*ALL'
session=''>
<callMethod name='myfunc' app='P42101'>
<params>
<param name='CostCtr'></param>
<param name='ExpDate'></param>
<param name='Quantity'></param>
</params>
</callMethod>
</jdeRequest>
```

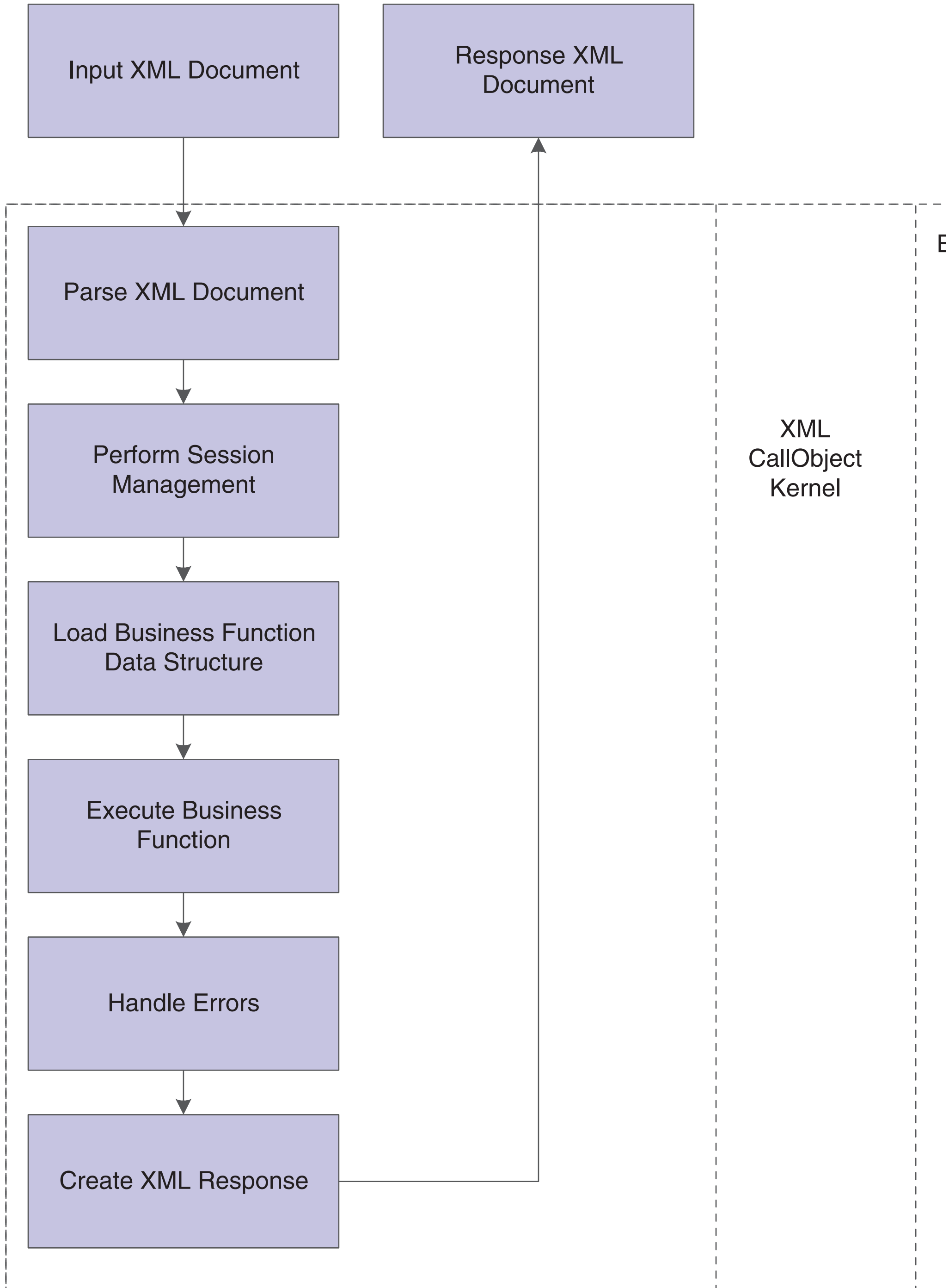
See XML Format Examples (All Parameters).

8.3 XML CallObject Process

This diagram illustrates XML CallObject processing:

Figure 8–1 XML CallObject process flow





In summary:

- The JD Edwards EnterpriseOne server receives an XML document.
- XML CallObject processes the message by parsing the XML document.
- The session manager validates the user and password.
- Each requested business function is called separately or within requested transaction boundaries until all calls are processed.

- Output data and error messages are merged with the data from the input XML document and a new response document is created and sent to the originator.

8.4 XML CallObject Document Format

This section provides an overview for formatting XML CallObject documents and discusses these elements:

- Call Object
- OnError Handling
- Call Object Error Handling
- Error Text
- Multiple Requests per Document
- ID/IDREF Support
- Return NULL Values

8.4.1 XML CallObject Formatting Documents

Your XML document must have these elements at the beginning of the document:

- jdeRequest Type
- Establish Session
- Expire Session

Your XML document must end with Terminate Session.

Your XML CallObject document can also have these optional elements:

- Call Object
- On Error Handling
- Call Object Error Handling
- Error Text
- Multiple Requests per Document
- ID/IDREF Support
- Return NULL Values

8.4.2 Call Object

Tags are used to call business functions on the server.

This sample code shows how to use callObject:

```
<?xml version='1.0' ?>
<jdeRequest type='callmethod' user='steve' pwd='xyz' role='*ALL'
environment='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 application attribute enables the business function to know who is calling it.

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

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

8.4.3 OnError Handling

You can add an onError element to the callMethod request to take a specific action if an error occurs. The onError tag can specify an abort attribute that specifies whether 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, which will be executed if errors were 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' role='*ALL'
environment='prod' session=''>
<callMethod name='myfunc' app='P42101' trans='t1' runOnError='yes'>
<params>
<param name='CostCtr'>    1001</param>
</params>
<onError abort='no'>
<endTransaction trans='t1' action='rollback'/>
</onError>
</callMethod>
</jdeRequest>
```

8.4.4 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' role='*ALL'
environment='prod' session=''>
<callMethod name='myfunc' app='P42101' trans='t1'>
<params>
<param name='CostCtr'>    1001</param>
</params>
<returnCode code='0'>Success</returnCode>
</callMethod>
</jdeResponse>
```

8.4.5 Error Text

Business function error messages are returned in the errors element. Within the errors element, there can be zero or more error elements that contain a code attribute for the error code and a child text node that contains the error text. The name attribute describes the parameter element that is referred to by the error.

```
<?xml version='1.0' ?>
<jdeResponse type='callmethod' user='steve' pwd='xyz' role='*ALL'
environment='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</error>
</errors>
</callMethod>
</jdeResponse>
```

8.4.6 Multiple Requests per Document

You can include multiple requests in the XML document. 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 you can override the default behavior by using the runOnError attribute on the request with a value of *yes*.

```
<?xml version='1.0' ?>
<jdeRequest type='callmethod' user='steve' pwd='xyz' role='*ALL'
environment='prod' session=''>
<callMethod name='myfunc' app='P42101' trans='t1' runOnError='yes'>
<params>
<param name='CostCtr'>    1001</param>
</params>
</callMethod>
</jdeRequest>
```

8.4.7 ID/IDREF Support

ID type attributes uniquely identify, by a string value, elements in a XML document. IDREF attributes enable 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 parameter element can specify an ID attribute so that its output value from the callMethod request will be saved and referred to later in another parameter element by an IDREF attribute. If a parameter element contains an IDREF attribute, the value of the given parameter is used as the input value for the parameter 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' role='*ALL'
environment='prod' session=''>
<callMethod name='myfunc' app='P42101' trans='t1' runOnError='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' runOnError='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 parameter elements. If the parameter elements contain IDREF attributes, then the referenced values are copied into the response.

8.4.8 Return NULL Values

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='' role='*ALL' 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>
```

8.5 XML CallObject jde.ini File Configuration

The XML CallObject kernel must be defined in the jde.ini file.

8.5.1 [JDENET_KERNEL_DEF6]

This examples illustrates settings for a Microsoft Windows platform:

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

This table provides the different .dll extensions for other platforms:

Platform	dispatchDLLName	dispatchDLLFunction
IBM i	XMLCALLOBJ	XMLCallObjectDispatch
HP9000	libxmlcallobj.sl	XMLCallObjectDispatch
SUN or RS6000	libxmlcallobj.so	XMLCallObjectDispatch

8.5.2 Example: CallObject Request

This code sample shows a CallObject request:

```
<?xml version="1.0" encoding="utf-8" ?>
<jdeRequest pwd="JDE" type="callmethod" user="JDE" role="*ALL"
 session="" environment="M7333NIS2" sessionidle="1800">
<callMethod app="XMLTest" name="AddressBookMasterMBF">
 <params>
  <param name="cActionCode">A</param>
  <param name="cUpdateMasterFile">1</param>
  <param name="mnAddressBookNumber" idref="ABNumber" />
```

```
  <param name="szSearchType">C</param>
  <param name="szAlphaName">bobs</param>
  <param name="szMailingName">Bob's Shrimp boats</param>
  <param name="szAddressLine1">One Technology Way</param>
  <param name="szPostalCode">80237</param>
  <param name="szCity">Denver</param>
  <param name="szCounty">Denver</param>
  <param name="szState">CO</param>
  <param name="szCountry">US</param>
  <param name="cPayablesYNM">N</param>
  <param name="cReceivablesYN">Y</param>
  <param name="cEmployeeYN">N</param>
  <param name="cUserCode">N</param>
  <param name="cARAPNettingY">N</param>
  <param name="jdDateEffective">01/23/2001</param>
  <param name="szProgramId">EP01012</param>
  <param name="mnAddNumParentOriginal">0</param>
  <param name="szVersionconsolidated" idref=Version />
  <param name="szCountryForPayroll">US</param>
 </params>
</callMethod>
</jdeRequest>
```

8.5.3 Example: CallObject Response

This code sample shows a CallObject response:

```
<?xml version="1.0" encoding="UTF-8" ?>
<jdeResponse pwd="JDE" role="*ALL" type="callmethod" user="JDE"
session="2360.1049473980.6" environment="PDEVNIS2" sessionidle="1800">
<callMethod app="XMLTest" name="AddressBookMasterMBF">
<returnCode code="0" />
 <params>
  <param name="cActionCode">A</param>
  <param name="cUpdateMasterFile">1</param>
  <param name="mnAddressBookNumber">57322</param>
  <param name="szSearchType">C</param>
  <param name="szAlphaName">bobs</param>
  <param name="szMailingName">Bob's Shrimp boats</param>
  <param name="szBusinessUnit">1</param>
  <param name="szAddressLine1">One Technology Way</param>
  <param name="szPostalCode">80237</param>
  <param name="szCity">Denver</param>
  <param name="szState">CO</param>
  <param name="szCountry">US</param>
  <param name="cPayablesYNM">N</param>
  <param name="cReceivablesYN">Y</param>
  <param name="cEmployeeYN">N</param>
  <param name="cUserCode">N</param>
  <param name="cARAPNettingY">N</param>
  <param name="cAddressType3YN">N</param>
  <param name="cAddressType4YN">N</param>
  <param name="cAddressType5YN">N</param>
  <param name="jdDateEffective"/>
  <param name="szProgramId">EP01012</param>
  <param name="szVersionconsolidated">ZJDE0001</param>
  <param name="cEdiSuccessfullyProcess">0</param>
  <param name="szCountryForPayroll">US</param>
 </params>
</callMethod>
```

```
</jdeResponse>
```

8.6 XML CallObject Return Codes

This table provides XML CallObject return codes that can be returned from ThinNet APIs:

Code	Description
0	XML request OK.
1	Root XML element is not a jdeRequest or jdeResponse.
2	The jdeRequest user identification is unknown. Check the user, password, and environment attributes. or A callmethod request is missing the session attribute.
3	An XML parse error exists at line.
4	A fatal XML parse exists error at line.
5	An error occurred during parser initialization; the server is not configured correctly.
6	There is an unknown parse error.
7	The request session attribute is invalid.
8	The request type attribute is invalid.
9	The request type attribute is not given.
10	The request session attribute is invalid; the referenced process 'processid' no longer exists.
11	The jdeRequest child element is invalid or unknown.
12	The environment 'Env name' could not be initialized for user. Check user, password, and environment attribute values.
13	The jdeXMLRequest parameter is invalid.
14	The connection to JD Edwards EnterpriseOne failed.
15	The jdeXMLRequest send failed.
16	The jdeXMLResponse receive failed.
17	The jdeXMLResponse memory allocation failed.
99	An invalid BSFN name exists.

9 Understanding XML Transaction

This chapter contains the following topics:

- Section 9.1, "XML Transaction"
- Section 9.2, "XML Transaction Update Process"
- Section 9.3, "XML Transaction Data Request"
- Section 9.4, "XML Transaction jde.ini File Configuration"

9.1 XML Transaction

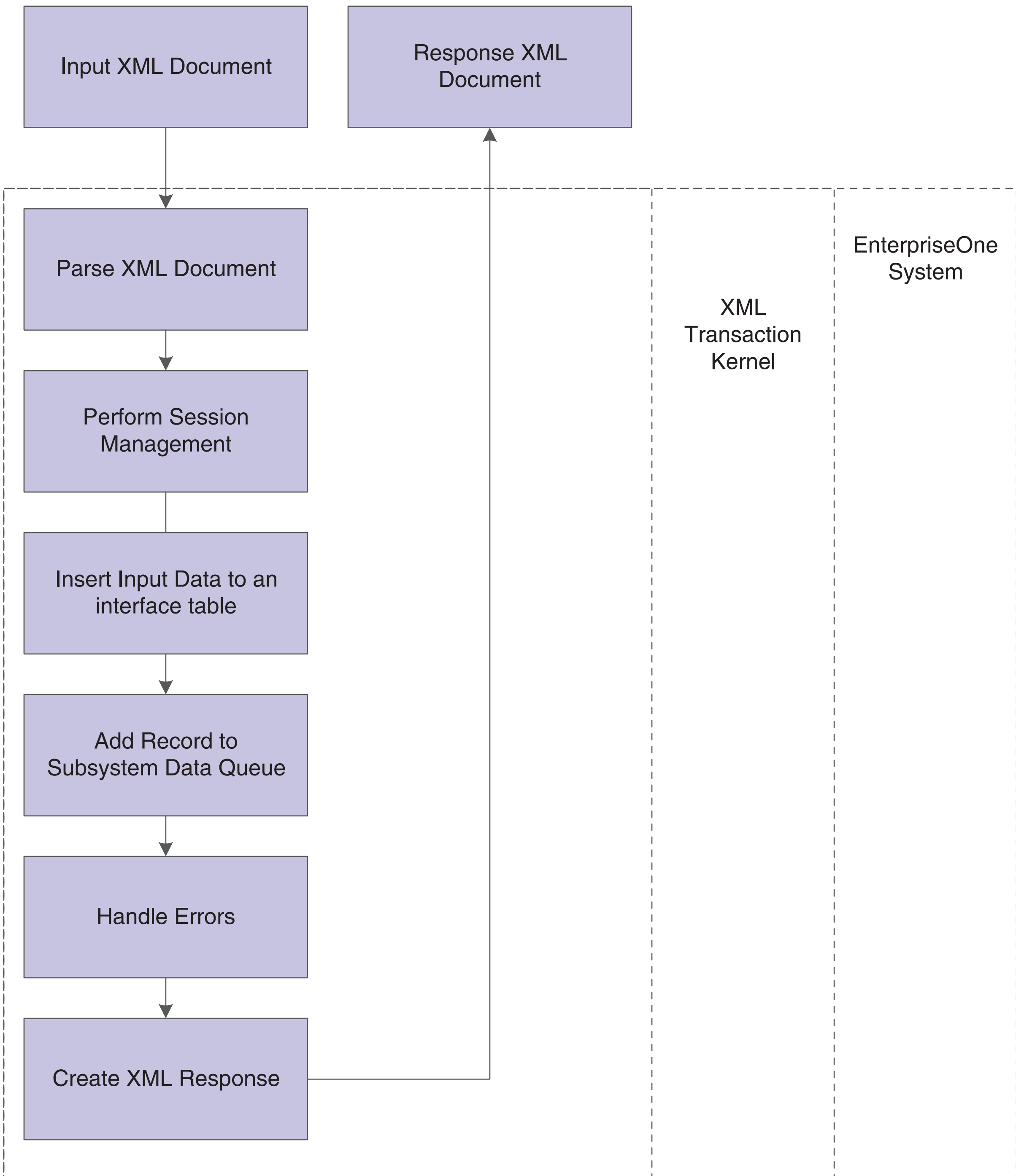
XML Transaction is XML-based interoperability that runs as a JD Edwards EnterpriseOne kernel process. You also can use XML Transaction with a messaging adapter. XML Transaction interacts with interface tables (Z tables) to update the database or to retrieve data. You can create one XML document that includes both updates to and retrieval of data from JD Edwards EnterpriseOne.

9.2 XML Transaction Update Process

To insert data into JD Edwards EnterpriseOne, you use a formatted XML document. The XML document includes a predefined transaction type, such as JDEOPIN. The XML document identifies one or more JD Edwards EnterpriseOne interface tables and lists all of the data (data type and actual data values) to be updated.

This illustration shows the XML Transaction update process.

Figure 9–1 XML Transaction data update process flow



In summary:

- A request in the form of an XML document contains a list of the data for a predefined transaction type.
- XML Transaction parses the XML inbound document and inserts the data into a JD Edwards EnterpriseOne inbound interface table.
- XML Transaction adds a subsystem data queue record to inform the JD Edwards EnterpriseOne subsystem to process the added record.

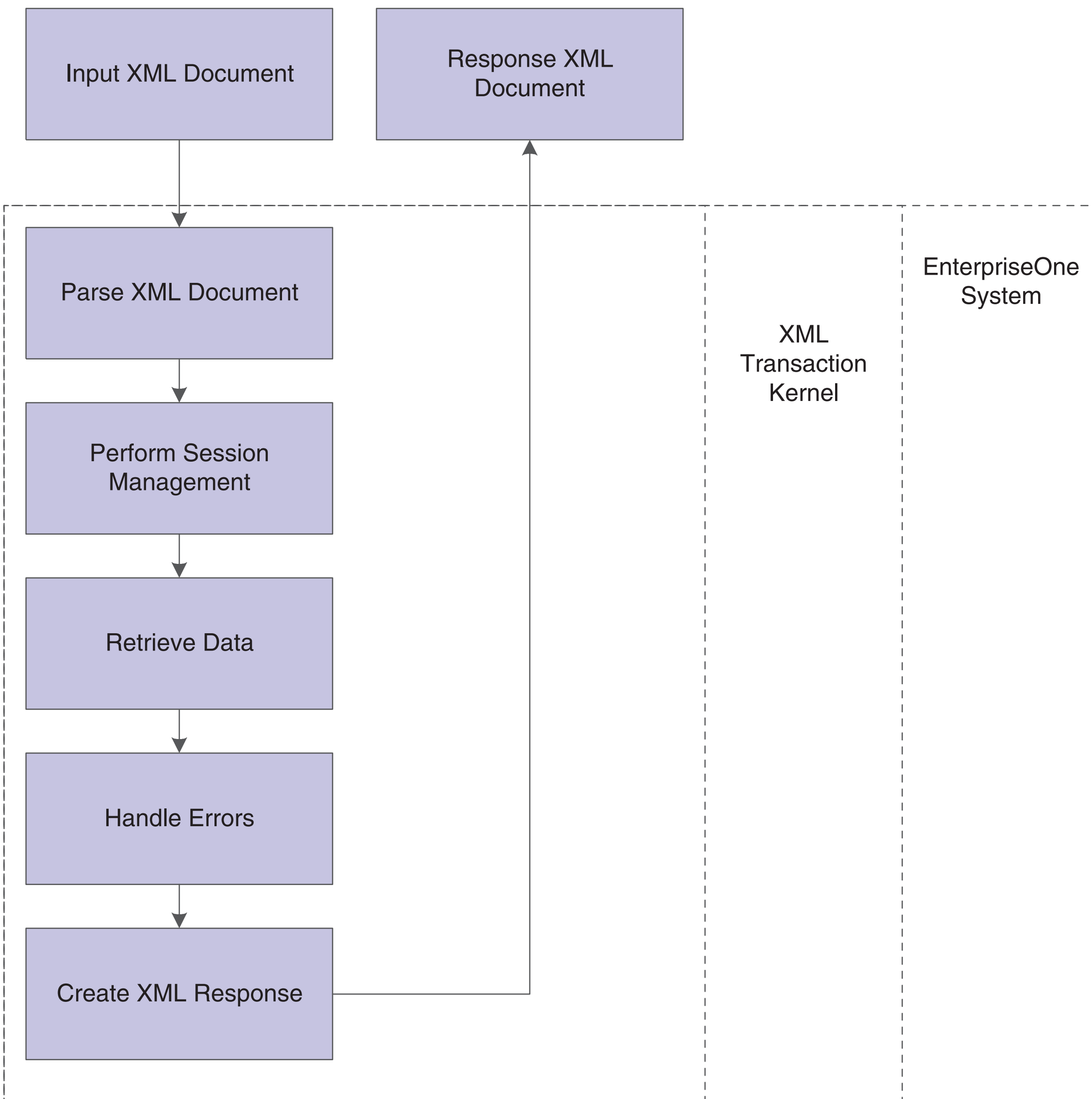
- The system sends a response to the requestor indicating whether the insertion into the interface table and the subsystem data queue addition were successful.

9.3 XML Transaction Data Request

To request data from JD Edwards EnterpriseOne, you use a formatted XML document. The XML document contains a transaction type, such as JDESOUT, and an index that identifies the data to be retrieved from the interface tables. You supply a template to retrieve the specific data.

This illustration shows the XML Transaction data request and response process:

Figure 9–2 XML Transaction data request process flow



In summary:

- A request in the form of an XML document contains the transaction type and an index of the requested data.
- XML Transaction parses the XML inbound document to get the transaction type and the index.
- XML Transaction retrieves the data from JD Edwards EnterpriseOne and inserts the data into interface tables.
- XML Transaction creates a response in the form of an XML document.

  The response is comprised of the interface table data records that match the transaction type and index. The response also contains any error messages that might have occurred.

9.4 XML Transaction jde.ini File Configuration

The XML Transaction kernel must be defined in the jde.ini file.

9.4.1 [JDENET_KERNEL_DEF15]

These settings are for a Microsoft Windows platform:

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

This table provides the different .dll extensions for other platforms:

Platform	dispatchDLLName	dispatchDLLFunction
IBM i	XMLTRANS	XMLTransactionDispatch
HP9000	libxmltransactions.sl	XMLTransactionDispatch
SUN or RS6000	libxmltransactions.so	XMLTransactionDispatch

9.4.2 Example: Outbound Order Status XML Request and Response Format

The XML transaction data request is created by the outbound function and sent to the XML transaction API. These code samples illustrate a sales order request and response.

The format in this XML Transaction request code sample returns all columns for the sales order header and detail lines:

```
<?xml version='1.0' ?>
<jdeRequest type='trans' user='user' pwd='password' environment='environment'
role='*ALL' session='' sessionidle='300'
<transaction action='transactionInfo' type='JDESOOUT'>
<key>
<column name='EdiUserId'>value</column>
<column name='EdiBatchNumber'>value</column>
<column name='EdiTransactNumber'>value</column>
</key>
</transaction>
</jdeRequest>
```

This code sample shows the XML Transaction response:

```
<?xml version='1.0' encoding='utf-8' ?>
<jdeResponse type='trans' user='user' role='*ALL' session='session1'
environment='env'>
 <transaction type='JDESOOUT' 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>
```

10

Understanding XML List

This chapter contains the following topics:

- Section 10.1, "XML List"
- Section 10.2, "List-Retrieval Engine Table Conversion Wrapper"
- Section 10.3, "XML List Process"
- Section 10.4, "XML List Requests"
- Section 10.5, "List-Retrieval Engine jde.ini File Configuration"
- Section 10.6, "XML List jde.ini File Configuration"

10.1 XML List

XML List is XML-based interoperability that runs as a JD Edwards EnterpriseOne kernel process. XML List provides List/GetNext functionality that enables you to collect a list of records from JD Edwards EnterpriseOne. XML List is built on the JD Edwards EnterpriseOne table conversion (TC) engine. XML List takes an XML document as a request and returns an XML document with the requested data. A list can represent data in a table, a business view, or data from a table conversion. Using data from a table conversion enables you to use multiple tables. By sending an XML document, you can retrieve metadata for a list, create a list, retrieve a chunk of data from a list, or delete a list. You can send the request through JDENet or third-party software to perform any of these operations:

- CreateList
- GetTemplate
- GetGroup
- DeleteList

XML List provides both trivial and non-trivial List/GetNext APIs. A trivial List/GetNext API performs simple gets such as selecting data from a single table. A non-trivial API uses additional functionality such as event rules. Each non-trivial List/GetNextBPAPI must have a table conversion designed for it. The data selection and data sequencing can be defined in an XML request at runtime.

XML List provides a list-retrieval engine that enables you to create an XML data file in the system repository and then retrieve the data in small chunks.

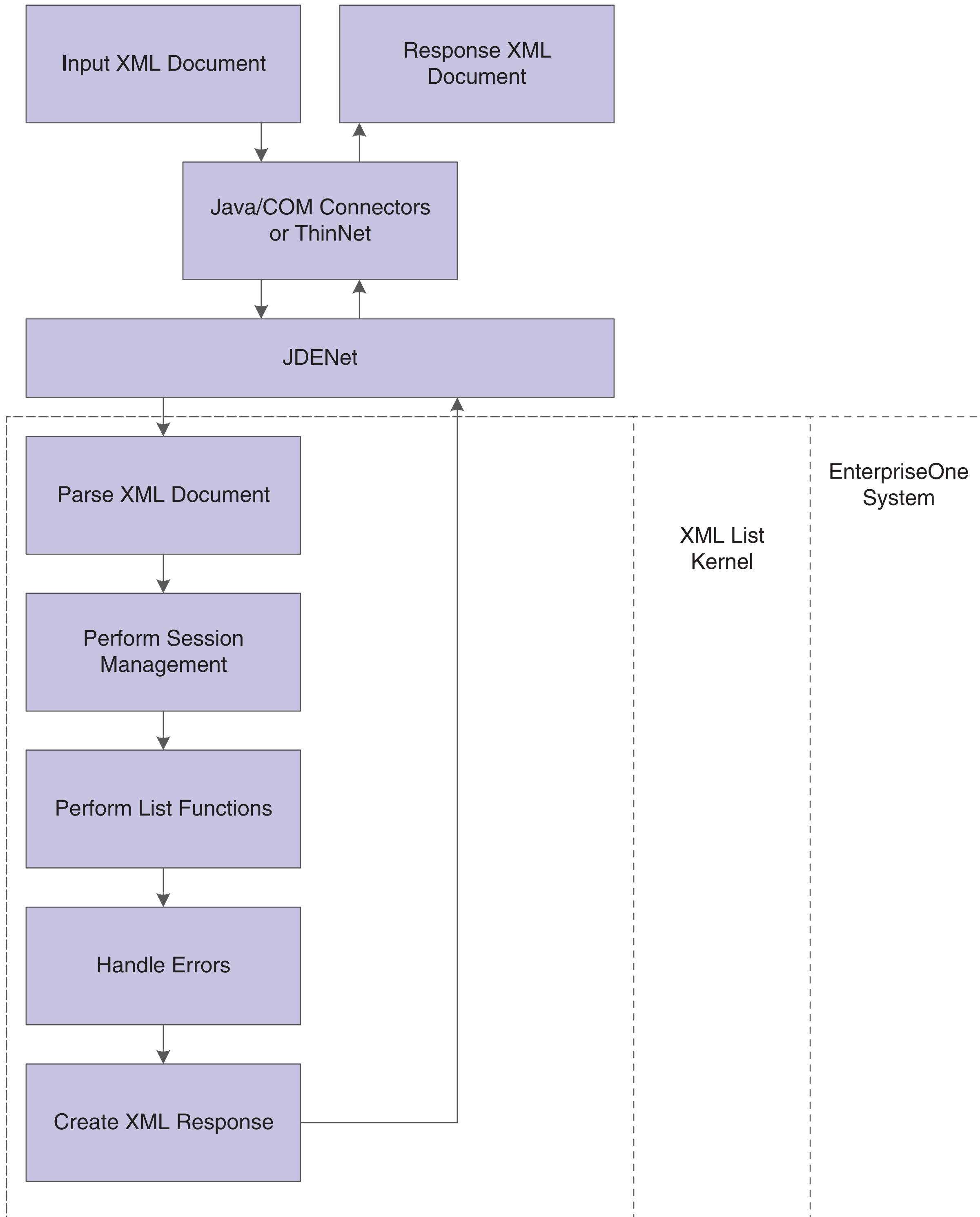
10.2 List-Retrieval Engine Table Conversion Wrapper

A list-retrieval engine is an optimized database engine that provides and manages access to XML repository files. Each XML list repository file is a pair of index and data files with *.idb and *.ddb extensions. The .idb file keeps an index that is generated on a data file, and the .ddb file keeps data that is generated by the table conversion engine. TCWrapper is a system module that aggregates list-retrieval and list-processing APIs from TCEngine and list-retrieval engine and provides a uniform access to the data for XML List.

10.3 XML List Process

This illustration shows the XML List process for both a trivial and non-trivial XML List request:

Figure 10–1 XML List process flow



In summary:

- JDENet receives the XML document.
- JDENet passes the XML document to the XML List kernel.
- If the request is for CreateList or GetTemplate, XML List creates a session.

- If the request is a trivial request, XML List retrieves the data and creates a response message to send to the requestor.
- If the request is a non-trivial request, XML List kernel passes the request to the appropriate API:
  - GetTemplate
  - CreateList
  - GetGroup
  - DeleteList
- A table conversion wrapper processes data retrieved as a result of a non-trivial request. The table conversion wrapper aggregates list-retrieval and list-processing APIs from the table conversion engine and the list-retrieval engine to provide a uniform access to the data.

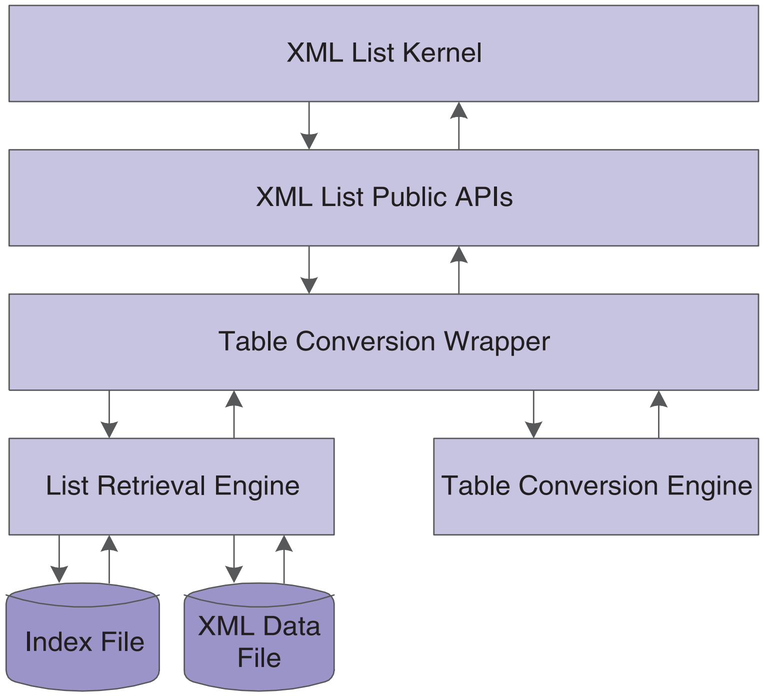
10.4 XML List Requests

You can make any of these requests using XML List:

XML List Request	Description
GetTemplate	Send a request to retrieve metadata information for a list so that you can add data selection and data sequencing to the CreateList request.
CreateList	Send a request with TC/Table name along with data selection and sequencing. The response is an XML document that has a handle and size that is associated with the created list in the repository.
GetGroup	Send a request to retrieve data from the generated list by the previous CreateList request. GetGroup passes the handle value and range of records to be retrieved.
DeleteList	Send a request to delete a list from the repository.

This illustration shows the various components in list operations:

Figure 10–2 XML List operations components



10.4.1 Creating a List

This code example illustrates using CreateList for an XML request with the TC Name/Table Name and data selection and sequencing. The system returns an XML response with a handle that is associated with the created list:

```
<?xml version="1.0"?>
<jdeRequest type="list" user="JDE" pwd="JDE" environment="PRODHP01"
role='*ALL' session="" sessionidle="">
 <ACTION TYPE="CreateList">
  <TC_NAME VALUE=""/>
  <TC_VERSION VALUE=""/>
  <FORMAT VALUE="UT"/>
    <RUNTIME_OPTIONS>
    <DATA_SELECTION>
     <CLAUSE TYPE="WHERE">
       <COLUMN NAME="" TABLE="" INSTANCE="" ALIAS=""/>
       <OPERATOR TYPE="EQ"/>
       <OPERAND>
        <COLUMN NAME="" TABLE="" INSTANCE="" ALIAS=""/>
        <LITERAL VALUE=""/>
        <LIST>
          <LITERAL VALUE=""/>
        </LIST>
        <RANGE>
          <LITERAL_FROM VALUE=""/>
          <LITERAL_TO VALUE=""/>
        </RANGE>
       </OPERAND>
     </CLAUSE>
<CLAUSE TYPE="OR">
       <COLUMN NAME="" TABLE="" INSTANCE="" ALIAS=""/>
       <OPERATOR TYPE="EQ"/>
```

```
        <OPERAND>
         <COLUMN NAME="" TABLE="" INSTANCE="" ALIAS=""/>
         <LITERAL VALUE=""/>
         <LIST>
           <LITERAL VALUE=""/>
         </LIST>
         <RANGE>
           <LITERAL_FROM VALUE=""/>
           <LITERAL_TO VALUE=""/>
         </RANGE>
        </OPERAND>
      </CLAUSE>
     </DATA_SELECTION>
     <DATA_SEQUENCING>
      <DATA SORT="ASCENDING">
        <COLUMN NAME="Product Code" TABLE="F0004" INSTANCE="" ALIAS=""/>
      </DATA>
     </DATA_SEQUENCING>
   </RUNTIME_OPTIONS>
  </ACTION>
</jdeRequest>
```

Either TC_NAME and TC_VERSION or TABLE_NAME and TABLE_TYPE must be defined in the request. TABLE_TYPE can be one of these:

- OWTABLE
- OWVIEW
- FOREIGN_TABLE

FORMAT VALUE is an optional attribute of the FORMAT element that enables full mode or concise mode formatting in the response message. UT is the only FORMAT value that is supported. If you do not set the VALUE="UT" attribute on the FORMAT element, the response message uses concise formatting, which is illustrated in this sample response:

```
<F0005>
   <SY>00</SY>
   <RT>03</RT>
   <KY>   DIR</KY>
   <DL01>Direct Manufacturing</DL01>
   <DL02>                  </DL02>
   <SPHD>     </SPHD>
   <UDCO></UDCO>
   <HRDC></HRDC>
   <USER>DEMO</USER>
   <PID>P00051</PID>
   <UPMJ>2055/05/12</UPMJ>
   <JOBN>V3477JG51</JOBN>
   <UPMT>175301</UPMT>
</F0005>
```

If you do not use the <FORMAT VALUE> element or you do not set the attribute to UT in the request, the response message uses full formatting, which is illustrated in this sample response:

```
<FORMAT NAME='F0005'>
    <COLUMN ALIAS='SY'>00</COLUMN>
    <COLUMN ALIAS='RT'>03</COLUMN>
    <COLUMN ALIAS='KY'>       DIR</COLUMN>
    <COLUMN ALIAS='DL01'>Direct Manufacturing</COLUMN>
    <COLUMN ALIAS='DL02'>                              </COLUMN>
```

```
    <COLUMN ALIAS='SPHD'>          </COLUMN>
    <COLUMN ALIAS='UDCO'> </COLUMN>
    <COLUMN ALIAS='HRDC'> </COLUMN>
    <COLUMN ALIAS='USER'>DEMO</COLUMN>
    <COLUMN ALIAS='PID'>P00051</COLUMN>
    <COLUMN ALIAS='UPMJ'>2055/05/12</COLUMN>
    <COLUMN ALIAS='JOBN'>V3477JG51</COLUMN>
    <COLUMN ALIAS='UPMT'>175301</COLUMN>
</FORMAT>
```

The CLAUSE can be WHERE, OR, or AND to simulate an SQL statement.

You can specify the COLUMN NAME with any meaningful name to help recognize the real column name in the table, which should be defined in ALIAS. The values of TABLE, INSTANCE, and ALIAS should be the same as those in the XML response that is returned by a GetTemplate request. For example, if Column X is in the data selection, it should be <COLUMN NAME=My column TABLE=F9999 INSTANCE=0 ALIAS=X/> because information is returned by a GetTemplate request and is similar to this example:

```
<COLUMN NAME="X" ALIAS="X" TYPE="String" LENGTH="32" TABLE="F9999" INSTANCE="0">
```

The OPERATOR uses values of EQ, NE, LT, GT, LE, GE, IN, NI, BW (between) or NB.

The OPERAND node can contain one of the these supported element types:

- Column
- Literal
- List
- Range

This XML node, which is a template fragment that should be used with only *one* of the supported elements, shows the supported elements in the OPERAND node (in bold type):

```
<CLAUSE TYPE="WHERE">
 <COLUMN NAME="UserDefinedCodes" TABLE="F0005" INSTANCE="" ALIAS="RT"/>
 <OPERATOR TYPE="EQ"/>
 <OPERAND>
    <COLUMN NAME="" TABLE="" INSTANCE="" ALIAS="null"/>
    <LITERAL VALUE="P4"/>
    <RANGE>
    </RANGE>
 </OPERAND>
</CLAUSE>
```

These sample XML nodes show the operator type and the operand using the different supported elements.

If the operand is a COLUMN, populate the COLUMN element. For example:

```
<CLAUSE TYPE="WHERE">
 <COLUMN NAME="DRSY" TABLE="F0005" INSTANCE="0" ALIAS="SY"/>
 <OPERATOR TYPE="EQ"/>
 <OPERAND>
    <COLUMN NAME="DRRT" TABLE="F0005" INSTANCE="0" ALIAS="RT"/>
 </OPERAND>
</CLAUSE>
```

If the operand is a LITERAL, populate the LITERAL element. For example:

```
<CLAUSE TYPE="WHERE">
 <COLUMN NAME="DRSY" TABLE="F0005" INSTANCE="0" ALIAS="SY"/>
 <OPERATOR TYPE="EQ"/>
 <OPERAND>
    <LITERAL VALUE="08"/>
 </OPERAND>
</CLAUSE>
```

If the operand is a LIST, populate the element LIST. LIST should be used with IN or NI. For example:

```
<CLAUSE TYPE="WHERE">
 <COLUMN NAME="DRSY" TABLE="F0005" INSTANCE="0" ALIAS="SY"/>
 <OPERATOR TYPE="IN"/>
 <OPERAND>
    <LIST>
        <LITERAL VALUE="08"/>
        <LITERAL VALUE="09"/>
    </LIST>
 </OPERAND>
</CLAUSE>
```

If the operand is a RANGE, populate the element RANGE. RANGE should be used with BW or NB. For example:

```
<CLAUSE TYPE="WHERE">
 <COLUMN NAME="DRSY" TABLE="F0005" INSTANCE="0" ALIAS="SY"/>
 <OPERATOR TYPE="BW"/>
 <OPERAND>
    <RANGE>
        <LITERAL_FROM VALUE="08"/>
        <LITERAL_TO VALUE="10"/>
    </RANGE>
 </OPERAND>
</CLAUSE>
```

The XML response for a CreateList request is similar to this:

```
<?xml version="1.0"?>
<jdeResponse type="list" session="5665.931961929.454">
<returnCode code="0">XMLRequest OK</returnCode>
 <ACTION TYPE="CreateList">
           <TABLE_NAME VALUE="F0005">
    <HANDLE>"1r4670001"</HANDLE>
    <SIZE>773</SIZE>
 </ACTION>
</jdeResponse>
```

The value of HANDLE can be published and referenced in a GetGroup or DeleteList request.

10.4.2 Retrieving Data from a List

You can retrieve data from a list generated by a previous CreateList request by using a GetGroup request. The HANDLE, FROM VALUE, and TO VALUE can be defined in the request:

```
<?xml version="1.0"?>
<jdeRequest type="list" user="JDE" pwd="JDE" role="*ALL" environment="PRODHP01">
 <ACTION TYPE="GetGroup">
  <HANDLE VALUE="1r4670001"/>
```

```
  <FROM VALUE="10"/>
  <TO VALUE="50"/>
 </ACTION>
</jdeRequest>
```

The XML response lists records falling into the range specified. The default FROM value is the first record and the default TO value is the last record in the list. For a GetGroup request for the whole list, no FROM and TO values need to be specified. In this sample code, the response returns the records in the list from #10 to #50:

```
<?xml version="1.0"?>
<jdeResponse type="list">
<returnCode code="0">XMLRequest OK</returnCode>
<ACTION TYPE="GetGroup">
<HANDLE VALUE="lr4670001"/>
   <FROM VALUE="10"/>
   <TO VALUE="50"/>
 <Format name="Standard"><Column name="X">abc</Column><Column name="Y">
edf</Column></Format>
 00
 </ACTION>
</jdeResponse>
```

10.4.3 Deleting a List

A list can be deleted if all GetGroup requests are done:

```
<?xml version="1.0"?>
<jdeRequest type="list" user="JDE" pwd="JDE" role="*ALL" environment="PRODHP01">
 <ACTION TYPE="DeleteList">
   <HANDLE VALUE="lr4670001"/>
 </ACTION>
</jdeRequest>
```

The list result defined in the HANDLE is deleted from the storage and a response with the status is returned to the caller:

```
<?xml version="1.0"?>
<jdeResponse type="list">
<returnCode code="0">XMLRequest OK</returnCode>
 <ACTION TYPE="DeleteList">
<HANDLE VALUE="lr4670001"/>
   <STATUS>OK</STATUS>
 </ACTION>
</jdeResponse>
```

10.4.4 Getting Column Information for a List

You can send a GetTemplate request to get the column information for a list so that data selection and sequencing can be added to the CreateList request. If OUTPUT is defined in the TEMPLATE_TYPE, the response is only for those columns in the XML output generated by a CreateList request based on the table conversion. For a trivial table conversion, both templates should be the same. The default template type is INPUT if no tag is specified.

```
<?xml version="1.0"?>
<jdeRequest type="list" user="JDE" pwd="JDE" role="*ALL"
environment="PRODHP01" session="" sessionidle="">
 <ACTION TYPE="GetTemplate">
  <TABLE_NAME VALUE="F0004"/>
```

```
  <TABLE_TYPE VALUE="OWTABLE"/>
  <TEMPLATE_TYPE VALUE="OUTPUT"/>
 </ACTION>
</jdeRequest>
```

The response for the input template lists all of the columns with alias name, type and the length of the data type, even though the length is only meaningful for the string type.

```
<?xml version="1.0"?>
<jdeResponse type="list" session="5665.931961929.454">
<returnCode code="0">XMLRequest OK</returnCode>
 <ACTION TYPE="GetTemplate">
  <TABLE_NAME VALUE="F0004"/>
  <TABLE_TYPE VALUE="OWTABLE"/>
  <TEMPLATE_TYPE VALUE="INPUT"/>
  <COLUMN Name="Address" Alias="X" TYPE="String" LENGTH="32" TABLE="F9999"
INSTANCE="0">
 </ACTION>
</jdeResponse>
```

10.5 List-Retrieval Engine jde.ini File Configuration

The list-retrieval engine uses a predefined folder as its system directory to keep and manage repository files. This system directory should be configured in jde.ini file as follows:

```
[LREngine]
System=C:\output
Repository_Size=20 (allocates percentage of disk free space for XML list
repository)
Disk_Monitor=Yes (monitors free space on the disk)
```

Note: The engine uses the IFS file system on IBM i, so a corresponding system subsection must be set up.

Caution: For data privacy, be sure to remove the global read access rights for the specified directory.

The [SECURITY] section of the jde.ini file should also be configured. The default environment, password, and user settings should be filled in for the engine to validate the default user and to initialize the default environment.

10.6 XML List jde.ini File Configuration

The XML List kernel must be defined in the jde.ini file.

10.6.1 [JDENET_KERNEL_DEF16]

Use these settings for a Microsoft Windows platform:

```
krnlName=XML LIST
dispatchDLLName=xmllist.dll
dispatchDLLFunction=_XMLListDispatch@28
maxNumberOfProcesses=3
```

```
beginningMsgTypeRange=5257
endingMsgTypeRange=5512
newProcessThresholdRequest=0
numberOfAutoStartProcesses=1
```

This table provides the different .dll extensions for other platforms:

Platform	dispatchDLLName	dispatchDLLFunction
IBM i	XMLLIST	XMLListDispatch
HP9000	libxmllist.sl	XMLListDispatch
SUN or RS6000	libxmllist.so	XMLListDispatch

11 Processing Z Transactions

This chapter contains the following topics:

- Section 11.1, "Understanding Z Transactions"
- Section 11.2, "Naming the Transaction"
- Section 11.3, "Adding Records to the Inbound Interface Table"
- Section 11.4, "Running an Update Process"
- Section 11.5, "Checking for Errors"
- Section 11.6, "Confirming the Update"
- Section 11.7, "Purging Data from the Interface Table"

11.1 Understanding Z Transactions

Z transactions are non-JD Edwards EnterpriseOne information that is properly formatted in the interface tables (Z tables) for updating to the JD Edwards EnterpriseOne database. Interface tables are working tables that mirror JD Edwards EnterpriseOne applications tables. JD Edwards EnterpriseOne provides predefined interface tables for some application transactions. You also can create your own interface tables as long as they are formatted according to JD Edwards EnterpriseOne standards.

You can process Z transactions into JD Edwards EnterpriseOne one transaction at a time (referred to as a batch of one), or you can place a large number of transactions into the interface table and then process all of the transactions at one time (referred to as a true batch).

See Also:

- Interoperability Interface Table Information.

11.2 Naming the Transaction

Z transaction types are defined in user-defined code 00/TT. If you create a new transaction, you must define the transaction in user-defined code 00/TT. When you name a new transaction type, the name must start with JDE and can be up to eight characters in length. These examples illustrate a proper transaction name:

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

11.3 Adding Records to the Inbound Interface Table

When you write your transaction to the appropriate interface table, you make the information available to JD Edwards EnterpriseOne for processing. Z transactions may be written directly to interface tables that are already in the EnterpriseOne database format. This list shows some of the ways that you can add records to the inbound interface tables:

- Create a flat file and then convert the flat file data into records in the interface table.

  See Understanding Flat Files.

- Write an Application Programming Interface (API) using JD Edwards EnterpriseOne-published APIs to update the interface table.

  See "API Fundamentals" in the *JD Edwards EnterpriseOne Tools Development Tools: APIs and Business Functions Guide*.

- Use Electronic Data Interchange (EDI) to update the interface table.

  See *JD Edwards EnterpriseOne Data Interface for Electronic Data Interchange 9.0 Implementation Guide*.

- Place a message in a WebSphere MQ or MSMQ messaging adapter.

  See JD Edwards EnterpriseOne and Messaging Queue Systems.

- Use Structured Query Language (SQL) or stored procedures. You must be able to convert your records to the JD Edwards EnterpriseOne interface table format.

> **Important:** If you are using a flat file to add records to the JD Edwards EnterpriseOne interface tables, verify that a version of the Inbound Flat File Conversion (R47002C) program exists for the transaction you are trying to create.

11.4 Running an Update Process

You can process Z transactions in one of these ways:

- Run an input batch process, which enables you to place a large number of transactions into the interface table and then process all of the transactions as one in batch mode.
- Run a subsystem job, which enables you to send transactions to JD Edwards EnterpriseOne one at a time without having to wait for completion to continue processing using the subsystem.

JD Edwards EnterpriseOne provides input batch and input subsystem processes for some applications.

11.4.1 Running an Input Batch Process

The input batch process enables you to place one or more records in an interface table and then run a UBE to process all of the records at one time. You initiate the input batch process for an application that supports inbound interoperability processing. When you select the input batch program, 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. You can change the processing options and data selection when you use a report version. The input batch process program generates an audit report that lists the transactions that were processed, totals for the number of processed transactions, and errors that occurred during processing.

11.4.2 Running a Subsystem Job

Subsystem jobs are continuous jobs that process records from a data queue and run until you terminate the job. Subsystem jobs read records one at a time for a subsystem table, retrieve information from that particular record, and run a UBE or table conversion for each record. This triggers the inbound processor batch process that processes that specific key. If required, a preprocessor runs from the inbound processor batch process to establish key information that matches the interface table record to the original application record (for example, the key to a cash receipt or purchase receipt). After processing the last record, instead of ending the job, subsystem jobs wait for a specific period and then attempt to retrieve a new record. For each subsystem job, multiple records can exist in the subsystem table.

You can schedule subsystem jobs.

You initiate a subsystem job in one of these ways:

Ways to Initiate Subsystem Jobs	Explanation
Use a business function	You can use the generic subsystem business function, Add Inbound Transaction to Subsystem Queue (B0000175), for inbound transactions. This function writes a record to the F986113 table to specify a batch process that needs to be awakened in the subsystem. The business function also passes keys to the subsystem data queue. The business function then starts processing the transaction.
Use the Solution Explorer	You can use the Solution Explorer to initiate the input subsystem batch process for an application that supports inbound interoperability processing. You start the subsystem job as you would a regular batch job. Unlike other batch jobs, subsystem jobs can only run on a server. Before processing, JD Edwards EnterpriseOne makes sure that limits for the subsystem job on the particular server have not been exceeded. If limits have been exceeded, the subsystem job will not be processed. To process your Z transaction in near real-time mode, start the subsystem when you start your system. You will need to place your request in the data queue before you write your transaction to the interface table.

Important: Instead of ending the job after the records have been processed, subsystem jobs look for new data in the data queue. Subsystem jobs run until you terminate them.

See Also:

- "Understanding the Scheduler Application" in the *JD Edwards EnterpriseOne Tools System Administration Guide*.
- "Understanding JD Edwards EnterpriseOne Subsystems" in the *JD Edwards EnterpriseOne Tools System Administration Guide*.

11.5 Checking for Errors

The input batch process uses the data in the interface tables to update the appropriate JD Edwards EnterpriseOne application tables as dictated by the business logic. If the process encounters an error for the transaction, the record is flagged in the processor audit trail report and error messages are sent to the employee work center in the form of action messages. These action messages, when invoked, call a revision application that enables you to make corrections to the interface table.

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 use a revision application to 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.

The system flags all transactions that have been successfully updated to the live files as successfully processed in the interface tables.

See Also:

- Using the Revision Application.

11.6 Confirming the Update

This step is optional. If you use a business function, you can provide a confirmation function to alert you that a transaction you sent into the JD Edwards EnterpriseOne system been processed. When processing is complete, JD Edwards EnterpriseOne calls the function that is specified in the request to notify you of the status of your process. The confirmation functions are written to your specifications, but you must use the JD Edwards EnterpriseOne defined data structure. Interoperability inbound confirmation functions are called from the inbound processor batch program through the Call Vendor-Specific Function - Inbound business function.

The confirmation function is specific to a process and must accept these parameters:

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

The first four parameters are the keys (EDUS, EDBT, EDTN, EDLN) to the processed transaction. The last 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.

After the subsystem batch process finishes processing the transaction, it calls the inbound confirmation function, passing the keys to the processed transaction and the notification about whether the transaction was successfully processed. 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 perform any of these tasks:

Task	Explanation
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 at an error status. Using the key returned to this function, you can access the transaction that is written to the interoperability interface table and retrieve any calculated or default information to update your original transaction.
Run other non-JD Edwards EnterpriseOne business processes	If your transaction is complete, you might want to run a business process that completes the transaction in the non-JD Edwards EnterpriseOne software.
Send messages to users	You might want to inform your users of the status of their original transactions.

11.7 Purging Data from the Interface Table

You should periodically purge records that have been successfully updated to the JD Edwards EnterpriseOne database from the interface tables.

See Also:

- Interoperability Interface Table Information.
- Purging Interface Table Information.

12 Using Flat Files

This chapter contains the following topics:

12.1 Understanding Flat Files

Flat files (also known as user-defined formats) are usually text files that are stored on your workstation or server and typically use the ASCII character set. Because data in a flat file is stored as one continuous string of information, flat files do not have relationships defined for them as relational database tables do. Flat files can be used to import or export data from applications that have no other means of interaction. For example, you might want to share information between JD Edwards EnterpriseOne and another system. If the non-JD Edwards EnterpriseOne system does not support the same databases that JD Edwards EnterpriseOne supports, then flat files might be the only way to transfer data between the two systems.

When you use flat files to transfer data to JD Edwards EnterpriseOne, the data must be converted to JD Edwards EnterpriseOne format before it can be updated to the live database. You can use JD Edwards EnterpriseOne interface tables along with a conversion program, electronic data interface (EDI), or table conversion to format the flat file data. You can use EDI or table conversion to retrieve JD Edwards EnterpriseOne data for input to a flat file.

WSG and some JD Edwards EnterpriseOne batch interfaces, such as the batch extraction programs, can accept flat files and parse the information to data format. Typically, WSGI uses the File I/O Adapter for flat file processing.

Note: JD Edwards EnterpriseOne supports flat file conversion on the Windows platform only.

See Also:

- JD Edwards EnterpriseOne Interface Tables.
- Interoperability Interface Table Information.
- "Setting Up Table Conversions" in the *JD Edwards EnterpriseOne Tools Development Tools: Data Access Tools Guide*.
- *JD Edwards EnterpriseOne Data Interface for Electronic Data Interchange 9.0 Implementation Guide*.

12.2 Formatting Flat Files

When you import data using JD Edwards EnterpriseOne interface tables, the format for flat files can be user-defined or character-delimited. This example illustrates a single database character record that has a user-defined format with five columns (Last, First, Addr (address), City, and Phone):

Last	First	Addr	City	Phone	Table Column Heading
Doe	John	123 Main	Any town	5551234	←database record

The user-defined format example is a fixed-width column format in which all of the data for each column starts in the same relative position in each row of data.

This is an example of the same data in a character-delimited format:

"Doe", "John", "123 Main", "Anytown", "5551234"

12.3 Setting Up Flat Files

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 the columns must appear in the same order as the interface table. Every field in the interface tables must be written to, even if the field is blank. Each field must be enclosed by a symbol that marks the start and end of the field. Typically, this symbol is a double quotation mark (“ “). In addition, each field must be separated from the next field with a field delimiter. Typically, this separator value is a coma (,). However, any field delimiter and text qualifier may be used as long as they do not interfere with the interpretation of the fields. You set the processing options on the conversion program to define the text qualifiers and field delimiters. If you are receiving documents with decimal numbers, you must use a placeholder (such as a period) to indicate the position of the decimal. You define the placeholder in the User Preference table.

The first field value in a flat file record indicates the record type. In other words, the first field value indicates into which interface table the conversion program should insert the record. Record type values are defined and stored by the record type user defined code 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

For example, suppose a record in the header table has this information (this example ignores table layout standards):

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

This is how the record in the flat file appears:

```
1, Joe,,Denver,80237
```

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

Dates must be in the format MM/DD/YY. Numeric fields must have a decimal as the place keeper. A comma cannot be used.

12.4 Converting Flat Files Using the Flat File Conversion Program

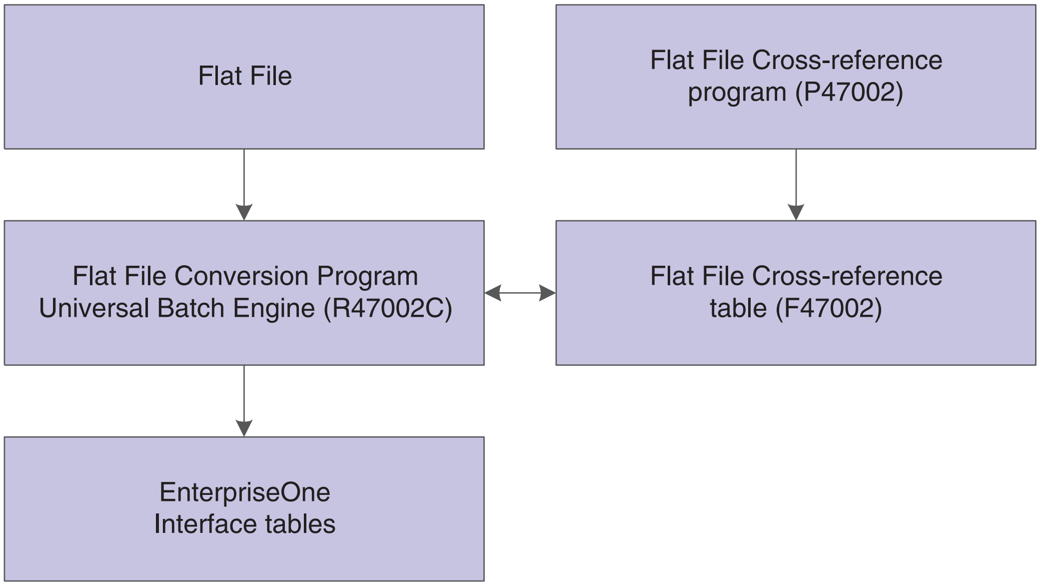
If you have a Windows platform, you can use the Inbound Flat File Conversion program (R74002C) or the Import Flat File To JDE File (B4700240) business function.

If you are on a Windows platform, you can use the Inbound Flat File Conversion program (R47002C) to import flat files into JD Edwards EnterpriseOne interface tables. You create a separate version of the Inbound Flat File Conversion program for each interface table.

> **Note:** To use the Inbound Flat File Conversion program, you must map a drive on your PC to the location of the flat file.

This diagram shows the process for updating JD Edwards EnterpriseOne interface tables using flat files:

Figure 12–1 Flat file conversion program process flow



You use the Flat File Cross-Reference program (P47002) to update the F47002 table. The conversion program uses the F47002 table to determine which flat file from which to read based on the transaction type that is being received. This list identifies some of the information that resides in the F47002 table:

- Transaction Type

  The specific transaction type. The transaction type must be defined in UDC 00/TT.

- Direction Indicator

  A code that indicates the direction of the transaction. The direction indicator code must be defined in UDC 00/DN.

- Flat File Name

  The path to the flat file on your Windows PC.

- Record Type

  An identifier that marks transaction records as header, detail, and so on. The record type indicator must be defined in UDC 00/RD.

- File Name

  A valid JD Edwards EnterpriseOne interface table.

The conversion program uses the Flat File Cross-Reference table to convert the flat file to the JD Edwards EnterpriseOne interface tables. The conversion program recognizes both the flat file it is reading from and the record type within that flat file. Each flat file contains records of differing lengths based on the corresponding interface table record.

The conversion program reads each record in the flat file and maps the record data into each field of the interface table based on the text qualifiers and field delimiters specified in the flat file. All fields must be correctly formatted for the conversion program to correctly interpret each field and move it to the corresponding field in the appropriate inbound interface table.

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, the interface table is not updated. Because the flat file is an external object that is created by third-party software, the conversion program is not able to determine which flat file data field is formatted incorrectly. You must determine what is wrong with the flat file. When the conversion program successfully converts all data from the flat file to the interface tables, the conversion program automatically deletes the flat file after the conversion. After the data is successfully converted and if you set the processing option to start the next process in the conversion program, the conversion program automatically runs the inbound processor batch process for that interface table. If you did not set up the processing option to start the inbound processor batch program, you must manually run the Flat File Conversion (R47002C) batch process.

If the flat file was not successfully processed, you can review the errors in the Employee Work Center, which you can access from the Workflow Management menu (G02). After you correct the error condition, run R47002C again.

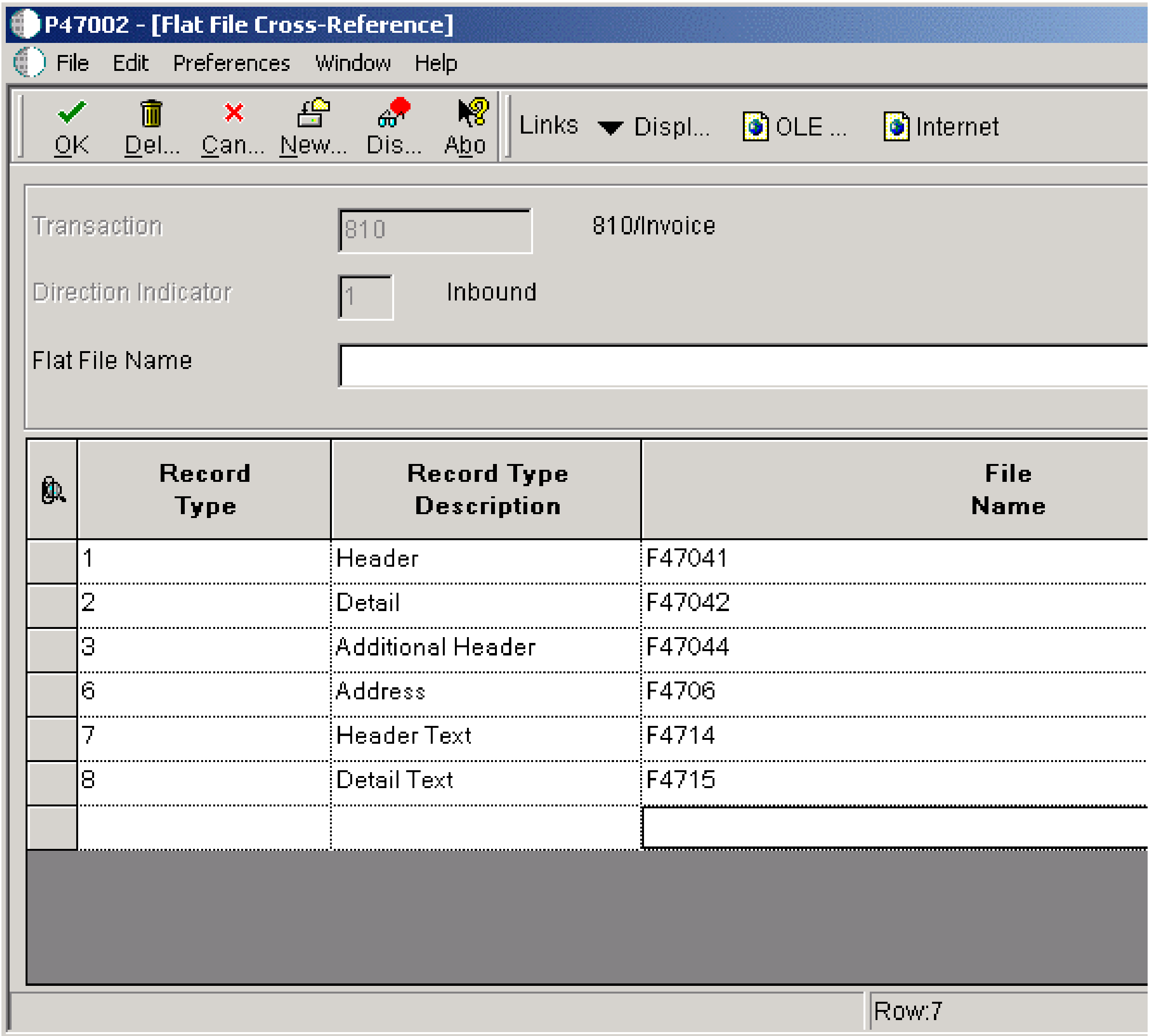
12.4.1 Forms Used to Convert Flat File Information

Form Name	FormID	Navigation	Usage
Work With Flat File Cross-Reference	W47002A	From an application that supports flat file conversion, open the Flat File Cross-Reference Program.	Identify the transaction type.
Flat File Cross-Reference	W47002B	On Work With Flat File Cross Reference, select the appropriate transaction in the detail area and then select Define from the Row menu.	Enter the name of the flat file, define the record types, and indicate the JD Edwards EnterpriseOne destination file.

12.4.2 Defining the Flat File Cross Reference Table

Access the Flat File Cross Reference form.

Figure 12–2 Flat File Cross Reference form



12.4.2.1 Flat File Cross Reference

Flat File Name

The name of the flat file. This includes the directory path where the flat file exists.

Record Type

The identifier that marks EDI transaction records as header and detail information. This is an EDI function only.

Record Type Description
A user-defined name or remark.

File Name
The number of a specific table. For example, the Account Master table name is F0901. See the Standards Manual on the programmers' menu for naming conventions.

12.5 Importing Flat Files Using a Business Function

If you are on a Windows platform, you can use the business function named Import Flat File To JDE File (B4700240). Because of changes to server operating systems and the various ways that operating systems store files, JD Edwards EnterpriseOne only supports the business function when run from a Windows platform. If you use the Import Flat File To JDE File (B4700240) business function, note these constraints:

- Transaction Type and Flat File Name fields must contain data.
- Only one character is allowed in the Record Type field.
- The maximum length per line is 4095 characters.
- The maximum record types are 40.
- Every line must have a record.
- The text qualifier cannot be the same as the column delimiter.

To ensure that flat file data is properly formatted before it is inserted into interface tables, the business function uses the F98713 table to obtain primary index key information. Normally, the F98713 table is located under the Default Business Data table mapping in the Object Configuration Manager. So that the business function can find the F98713 table, you must take one of these actions:

- Map the F98713 table in the system data source.
- Ensure the F98713 table exists in the business data source.

12.5.1 Map the F98713 table in the System Data Source

To map the table in the system data source, add an OCM mapping that points the F98713 table to the central objects data source.

12.5.2 Ensure the F98713 table Exists in the Business Data Source

If you generate the F98713 table in the business data source, you must ensure that file extensions on your PC are hidden. To hide file extensions, complete these steps:

1. From Start/Settings/Control Panel/Folder Options, click the View tab.
2. Select the Hide file extension for known file types option, and then click OK.

You must also ensure that the Flat File Name field in the F47002 table has a file extension. For example: C:\flatfiles\850.txt.

12.5.3 Flat File Conversion Error Messages

These two errors might occur when you use the business function to convert flat files:

- 4363 Null Pointer
- 4377 Invalid Input Parameter

Both of the errors are internal problems within the business function.

These errors might occur as a result of problems with user setup or with the configurable network computing (CNC) implementation:

- 0073 Invalid File Name
- 128J (filename) Insert Failed
- 3003 Open of File Unsuccessful
- 4569 Invalid Format

12.6 Converting Flat Files Using APIs

In addition to the existing flat file APIs, JD Edwards EnterpriseOne provides APIs for non-Unicode flat files. The Unicode APIs are required when flat file data is written to or read by a process outside of JD Edwards EnterpriseOne. The JD Edwards EnterpriseOne APIs, such as jdeFWrite() and jdeFRead(), do not convert flat file data, which means that the default flat file I/O for character data is in Unicode. If you use JD Edwards EnterpriseOne-generated flat files and the recipient system is not expecting Unicode data, you will not be able to read the flat file correctly. For example, if the recipient system is not Unicode enabled and the system is expecting data in the Japanese Shift_JIS code page (or encoding), you will not be able to read the flat file correctly. To enable the creation of the flat file in the Japanese Shift_JIS page, the application that creates the flat file must be configured using the Unicode Flat File Encoding Configuration program (P93081). If the flat file is a work file or debugging file and will be written and read by JD Edwards EnterpriseOne only, the existing flat file APIs should be used. For example, if the business function is doing some sort of caching in a flat file, that flat file data does not need to be converted.

The JD Edwards EnterpriseOne conversion to Unicode uses UCS-2 encoding in memory, or two bytes per character (JCHAR), for representation of all character data. The character data that is passed to the output flat file APIs needs to be in JCHAR (UCS-2). The input flat file APIs converts the character data from a configured code page to UCS-2 and returns the character in JCHAR (or JCHAR string). The flat file conversion APIs enable you to configure a code page for the flat file at runtime. You use P93081 to set up the flat file code page. Flat file encoding is based on attributes such as application name, application version name, user name, and environment name.

If no code page is specified in the configuration application, the APIs perform flat file I/O passing through the data as it was input to the specific function. For example, jdeFWriteConvert() writes Unicode data and no conversion is performed.

See Also:

- "Understanding Foreign Tables" in the *JD Edwards EnterpriseOne Tools Development Tools: Data Access Tools Guide*.

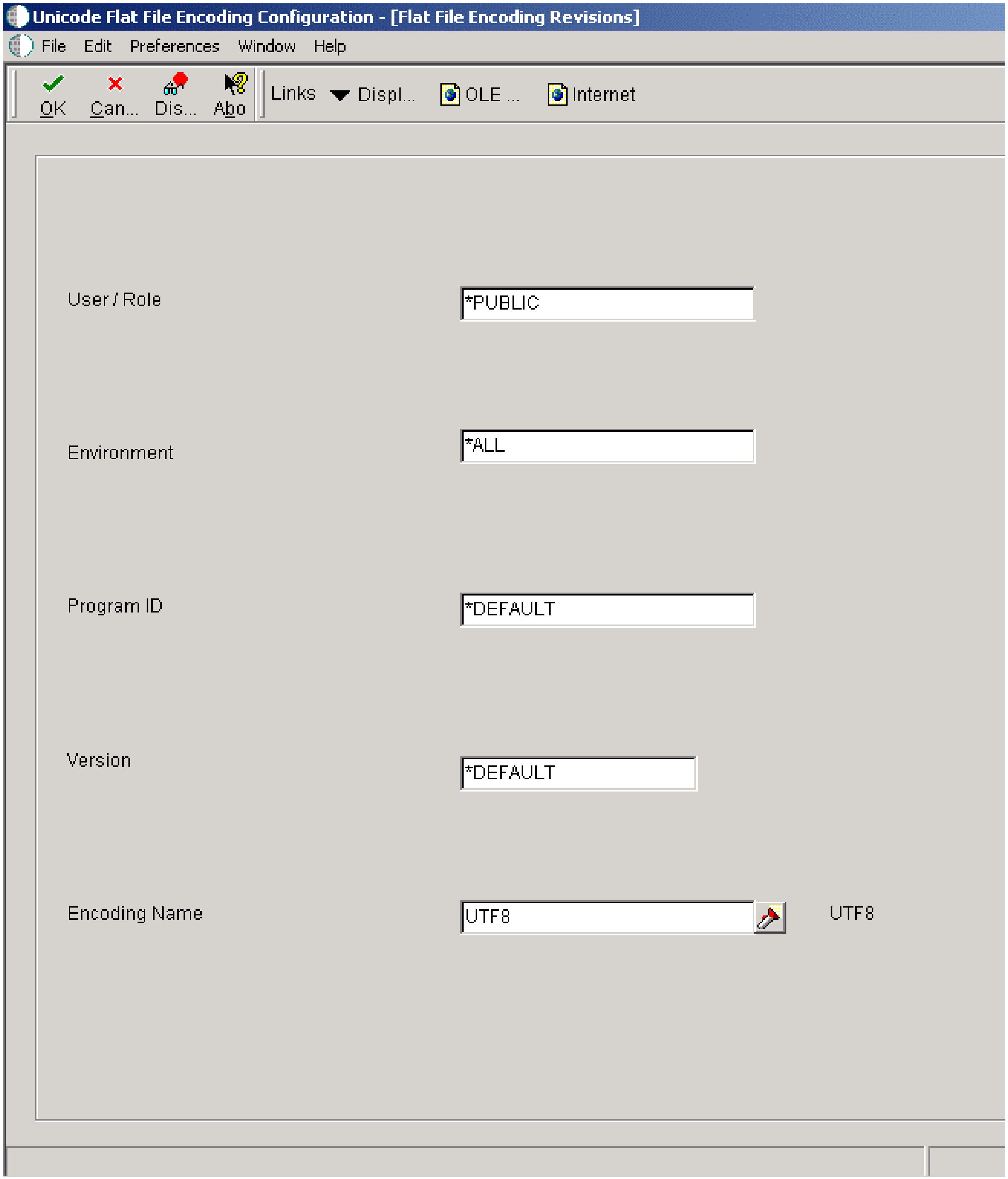
12.6.1 Forms Used to Convert Flat File Information

Form Name	FormID	Navigation	Usage
Work With Flat File Encoding	W93081A	From the Windows client, select System Administration Tools (GH9011), System Administration Tools, User Management, User Management Advanced and Technical Operations, Unicode Flat File Encoding Configuration	Locate and review existing Unicode flat file encoding configurations.
Flat File Encoding Revision	W93081B	On Work With Flat File Encoding, click Add	Add or change Unicode flat file encoding configuration information.
Work With Flat File Encoding	W93081A	On Work With Flat File Encoding, click Find, select your newly added Unicode configuration record in the detail area, and then select Change Status from the Row menu.	Activate or deactivate a Unicode configuration record.

12.6.2 Setting Up Flat File Encoding

Access the Unicode Flat File Encoding Configuration form from the Windows client.

Figure 12–3 Unicode Flat File Encoding Configuration form


Unicode Flat File Encoding Configuration - [Flat File Encoding Revisions]
File Edit Preferences Window Help
OK Can... Dis... Abo Links Displ... OLE ... Internet
User / Role
*PUBLIC
Environment
*ALL
Program ID
*DEFAULT
Version
*DEFAULT
Encoding Name
UTF8
UTF8

12.6.2.1 Flat File Encoding Configuration

User / Role
A profile that classifies users into groups for system security purposes. You use group profiles to give the members of a group access to specific programs.

Some rules for creating a profile for a user class or group include:

- The name of the user class or group must begin with an asterisk (*) so that it does not conflict with any system profiles.
- The User Class/Group field must be blank when you enter a new group profile.

Environment
For install applications, the environment name is also called the Plan Name and uniquely identifies an upgrade environment for install/reinstall.

For environment or version applications, this is the path code that identifies the location of the application or version specification data.

Program ID
The number that identifies the batch or interactive program (batch or interactive object). For example, the number of the Sales Order Entry interactive program is P4210, and the number of the Print Invoices batch process report is R42565.

The program ID is a variable length value. It is assigned according to a structured syntax in the form TSSXXX, where:

- T is an alphabetic character and identifies the type, such as P for Program, R for Report, and so on.

  For example, the value P in the number P4210 indicates that the object is a program.
- SS are numeric characters and identify the system code.

  For example, the value 42 in the number P4210 indicates that this program belongs to system 42, which is the Sales Order Processing system.
- XXX (the remaining characters) are numeric and identify a unique program or report.

  For example, the value 10 in the number P4210 indicates that this is the Sales Order Entry program.

Version
A user-defined set of specifications that control how applications and reports run. You use versions to group and save a set of user-defined processing option values and data selection and sequencing options. Interactive versions are associated with applications (usually as a menu selection). Batch versions are associated with batch jobs or reports. To run a batch process, you must select a version.

Encoding Name
A code that indicates the name of the encoding that the system uses to produce or consume flat files.

13 Understanding Messaging Queue Adapters

This chapter contains the following topics:

13.1 JD Edwards EnterpriseOne and Messaging Queue Systems

JD Edwards EnterpriseOne supports both Microsoft and IBM message queueing systems. If your system can implement the messaging protocols and produce and consume XML documents using the formats discussed in this document, you can use a messaging queue adapter to send information to and receive information from JD Edwards EnterpriseOne. The messaging adapters for JD Edwards EnterpriseOne are Oracle products that can be licensed and installed independently from JD Edwards EnterpriseOne.

13.2 Data Exchange Between JD Edwards EnterpriseOne and a Messaging Queue Adapter

The JD Edwards EnterpriseOne messaging adapters, adapter for MSMQ and adapter for WebSphere MQ, enable you to connect any third-party application to JD Edwards EnterpriseOne for sending and receiving messages. The messaging adapter monitors an inbound queue for request and reply messages, performs the requested services, and places the results on outbound queues. The messaging adapter also monitors JD Edwards EnterpriseOne for specified activities and then publishes the results in an outbound message queue. All messages transported through the messaging system are in the form of XML documents. The required elements for formatting XML documents are discussed in the Using XML chapter.

See Formatting XML Documents.

13.2.1 Sending Information to JD Edwards EnterpriseOne

Third-party applications can send information to JD Edwards EnterpriseOne. These inbound transactions are called Z transactions. XML CallObject is used for processing Z transactions. The XML CallObject process flow, jde.ini file configuration, and elements specific to XML CallObject formatting are discussed in the XML CallObject chapter.

See XML CallObject.

13.2.1.1 Z Transaction Process Flow

A typical flow for processing Z transactions is:

- The adapter picks up a message in XML format from the message queue.
- The XML document is passed into the jdeXMLCallObject Application Programming Interface (API).
- The session manager validates user and password.
- The JD Edwards EnterpriseOne server processes the message by parsing the XML document.
- Each requested business function is called separately or within requested transaction boundaries until all calls are processed.
- Transactions are added to the JD Edwards EnterpriseOne database.
- Output data and error messages are merged back into the XML document and a new response document is created.
- The adapter places the response XML document in the queue.

  The response can be an error or success XML document.

See Understanding Z Transactions.

13.2.2 Retrieving Information from JD Edwards EnterpriseOne

Third-party applications can retrieve information from JD Edwards EnterpriseOne. These outbound transactions are called events. You can use a message queuing system (MSMQ or WebSphere MQ) to receive events. The messaging queue adapter provides a layer over existing functionality. JD Edwards EnterpriseOne supports these three kinds of events:

- Real-time events
- XAPI events
- Z events

To receive guaranteed real-time and XAPI events, you must set up a real-time event queue. In addition, you must set up your events and configure your system to receive guaranteed events. The Using Guaranteed Events chapter discusses how the system processes events and provides information for configuring your system to receive guaranteed events. The Realtime Events Reference guide provides information for creating real-time and XAPI events. You can create custom XML documents. To create custom XML documents, you can find or create a business function to accomplish the required task, or you can retrieve an XML template.

See XML Transaction.

See XML Format Examples (Events).

See Understanding Guaranteed Events.

See Creating MSMQ Queues.

See Creating WebSphere MQ Queues.

See *JD Edwards EnterpriseOne Application Real Time Events 9.0 Guide*.

13.2.3 Using JD Edwards Classic Event System

You can use the JD Edwards EnterpriseOne Classic Event Delivery system to receive reliable real-time, XAPI, and Z events. The JD Edwards EnterpriseOne system includes an Event Notification (EVN) Kernel that manages subscribers and notifies them when an event (Z, real-time, or XAPI) occurs. The EVN Kernel can distribute events through WebSphere MQ or MSMQ transport drivers to the messaging queue system. You must set up the system so that the appropriate event type is generated. Real-time and XAPI events must be defined in the F90701 table.

Use the setup features described in the Classic Events chapters of this guide and the Real-Time Events Reference guide to receive reliable real-time, XAPI, and Z events. The sample code for requests and responses, and jde.ini file configuration are discussed in detail in the XML Transaction chapter.

See Understanding XML.

See XML Transaction.

See Understanding Classic Events.

See Understanding Real-Time Events - Classic.

See Understanding XAPI Events - Classic.

See Understanding Z Events - Classic.

See *JD Edwards EnterpriseOne Application Real Time Events 9.0 Guide*.

13.2.3.1 Classic Z Event Processing

A typical flow for processing classic Z events is:

- 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.
- The outbound notification function sends a net message, and the kernel picks up the message and calls the outbound function for the event type.
- The messaging adapter reads the message and calls the appropriate API.
- The adapter uses the record key from the JDENET message.
- An XML response document is created.
- The XML document is placed in the outbound queue.

13.2.3.2 Enabling Z Events Interface Table Processes

To send JD Edwards EnterpriseOne transactions to a messaging queue system such as IBM's WebSphere MQ or Microsoft's Message Queuing systems, you can use JD Edwards EnterpriseOne Z event functionality. An interface table (also called Z table) is a working table where data is collected for sending to a third-party application or system.

13.2.3.3 Outbound Table Adapter Function

You use the OutboundZTableAdapter function to send a message from an outbound interface table to a messaging adapter queue. The function is invoked from the kernel dispatch function, which then sends the net message data that contains the parameters from the interface table subsystem Universal Batch Engine (UBE). This example shows the outbound table adapter function:

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

The parameters define the records and the transaction type to be used to cross-reference the tables that contain the data to populate the message that is sent to the message adapter queue. The messaging-specific OutboundZTableAdapter parses the net message data and calls the XML Interface Table Inquiry API to fetch the records from the interface table and format the results into an XML string.

You must set up JD Edwards EnterpriseOne to initiate the outbound interface table process. The format of the outbound interface table message has an XML based format.

13.2.3.4 Outbound Notification

The outbound notification function is called by the standard generic Outbound Subsystem batch process UBE and provides notification that records have been placed in the interface tables.

This function passes the key fields for a record in the JD Edwards EnterpriseOne Outbound Transaction interface tables to the outbound adapter. With these key fields, you can process the information from the database record into a message queue. This example shows an outbound notification message:

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

This list provides the required input parameters:

- 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 in a JDENET message:

- Environment Name - use JD Edwards EnterpriseOne APIs to retrieve environment from the subsystem batch process.
- User ID - key to interface table record.
- Batch Number - key to interface table record.
- Transaction Number - key to interface table record.
- Line Number - key to interface table record.
- Transaction Type - tie to an interface table.
- Document Type - (optional).
- Sequence Number - (optional).

The key information in the JDENET message packets is used by the outbound adapter to retrieve the record from the interface table. The transaction type enables the adapter to be generic and enables the adapter to process other transactions in the future. The transaction type maps to the F47002 table to determine the interface tables.

13.2.4 XML Interface Table Inquiry API

The XML interface table inquiry API (jdeRetrieveTransactionInfo) receives an XML string that includes the table record key and returns an XML string for outbound processing.

The messaging adapter calls the API. The API parses the XML string. Based on the transaction type, the API goes to the F47002 table to determine from which interface to fetch records. The F47002 table has a record for each table associated with the transaction type. Using JDB database APIs, XML Interface Table Inquiry then uses the index found in the XML string to fetch records from the interface table and returns the results in an XML string.

13.3 Management of the Messaging Queue Adapter Queues

The messaging adapters accept input and produce output by reading and writing to messaging queues. You create specific queues for the messaging adapter to use. You must specify the names of these queues in the jde.ini file on the JD Edwards EnterpriseOne server so that the messaging adapter can find them. The adapter configuration specifications are defined within the jde.ini initialization file that is read upon startup of the JD Edwards EnterpriseOne server. Typically, the system administrator configures the jde.ini file settings, but you might need to change the settings or verify that the settings are correct.

When you install a message adapter, you are asked to create several message queues. This table lists the queues and platforms that reside on the JD Edwards EnterpriseOne server and provides recommended names based on the platform:

Queue	MSMQ Platform and Recommended Name	IBM i Platform and Recommended Name	NT Platform and Recommended Name	UNIX Platform and Recommended Name
Inbound	<computer name>\inbound	INBOUND.Q	INBOUND.Q	INBOUND.Q
Outbound	<computer name>\outbound	OUTBOUND.Q.XMIT	OUTBOUND.Q.XMIT	OUTBOUND.Q.XMIT
Success	Not applicable	SUCCESS.Q	SUCCESS.Q	SUCCESS.Q
Error	<computer name>\error	ERROR.Q	ERROR.Q	ERROR.Q
Default Response	Not applicable	DEFRES.Q	DEFRES.Q	DEFRES.Q

Important: Queue names for IBM Websphere Message Queue must be all upper case.

Note: The queue names in the jde.ini file must correspond to the queue names on the server.

13.3.1 Inbound Queue

The inbound queue stores all inbound messages to JD Edwards EnterpriseOne. After the message is processed, it is removed from the queue. The install suggests calling the queue INBOUND.Q. You must specify the queue name in the QInboundName setting in the jde.ini file.

13.3.2 Outbound Queue

The outbound queue stores the outbound messages from JD Edwards EnterpriseOne. The install suggests calling the queue OUTBOUND.Q. You must specify the queue name in the QOutboundName setting in the jde.ini file.

13.3.3 Success Queue

The success queue stores successfully processed messages in JD Edwards EnterpriseOne. These messages contain return code information for the business function calls and default or calculated parameter information. The messages remain in the queue until you remove them. The install suggests calling the queue SUCCESS.Q. You must specify the queue name in the XML document within the returnParms tag. If you do not specify a success destination queue within the XML document and you leave the QErrorName blank in the jde.ini, the message is not written to any queue.

13.3.4 Error Queue

The error queue stores processed messages that are in error in JD Edwards EnterpriseOne. These messages contain return code information for the business function calls, default and calculated parameter information, and error information. These messages remain in the queue until you remove them. The install suggests calling the queue ERROR.Q. You must specify the queue name in the XML document within the returnParms tag. If you do not specify a failure destination queue within the XML document and you leave the QErrorName blank in the jde.ini, the message is not written to any queue.

13.3.5 Default Response Queue

The default response queue stores the processed messages into JD Edwards EnterpriseOne. These messages may be in error or successfully processed. The messages contain return code information for the business function calls, default or calculated parameter information, and possibly error information. These messages remain in the queue until you remove them. The install suggests calling the queue DEFRES.Q. You must specify the queue name in the QErrorName setting in the jde.ini file. If you do not specify a success or failure destination queue in the XML document, the queue you set in the jde.ini file is used as the default queue for the message. If the QErrorName setting is also blank, the message is not written to any queue.

Note: The commands for creating these queues along with a discussion of other queues are provided in the applicable configuration document.

13.4 Configuration of the jde.ini File to Support Messaging Queue Adapters

The JD Edwards EnterpriseOne messaging adapters use settings in the MQSI section (for IBM) or the MSMQ section (for Microsoft) of the jde.ini file to start, to monitor queues, and to send error messages. The names of queues are case-sensitive. The jde.ini file can be modified for messaging queues and for JD Edwards EnterpriseOne UBE queues. Refer to the appropriate Messaging Adapter Installation documentation for more information about setting up queues and the jde.ini file settings. The queue names you use must correspond with the queue names you have set up on the server.

14

Using Guaranteed Events

This chapter contains the following topics:

Note: This chapter is applicable only if you use guaranteed events delivery. Guaranteed event delivery is available when you use JD Edwards EnterpriseOne Tools 8.94 with JD Edwards EnterpriseOne Applications 8.11 or JD Edwards EnterpriseOne Tools 8.95 and later Tools releases with JD Edwards EnterpriseOne Applications 8.10, 8.11, and later Applications releases.

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

14.1 Understanding Guaranteed Events

Oracle JD Edwards EnterpriseOne event functionality provides an infrastructure that can capture JD Edwards EnterpriseOne transactions in various ways and provide real-time notification to third-party software, end users, and other Oracle systems, such as Web Services Gateway (WSG) and Customer Relationship Management (CRM).

JD Edwards EnterpriseOne notifications are called events. The JD Edwards EnterpriseOne event system implements a publish and subscribe model. Events are delivered to subscribers in XML documents that contain detailed information about the event. For example, when a sales order is entered into the system, the sales order information can be automatically sent to a CRM or supply chain management (SCM) application for further processing. If your system is IBM, you can use the WebSphere MQ messaging system to receive events. If your system is Microsoft, you can use the MSMQ messaging system to receive events. WebSphere MQ and MSMQ provide a point-to-point interface with JD Edwards EnterpriseOne.

JD Edwards EnterpriseOne supports these three kinds of events:

Event Category	Purpose	Generation Mechanism	Response Capability
Real-Time Event	Provides requested notification to third-party software, end-users, and other Oracle systems when certain transactions occur.	System calls	No
XAPI Event	Provides requested notification to third-party software, end-users, and other Oracle systems when certain transactions occur and to provide a response.	System calls	Yes
Z Event	Provides requested notification to third-party software, end-users, and other Oracle systems when certain transactions occur.	Interface tables and system calls	No

14.2 Processing Guaranteed Events

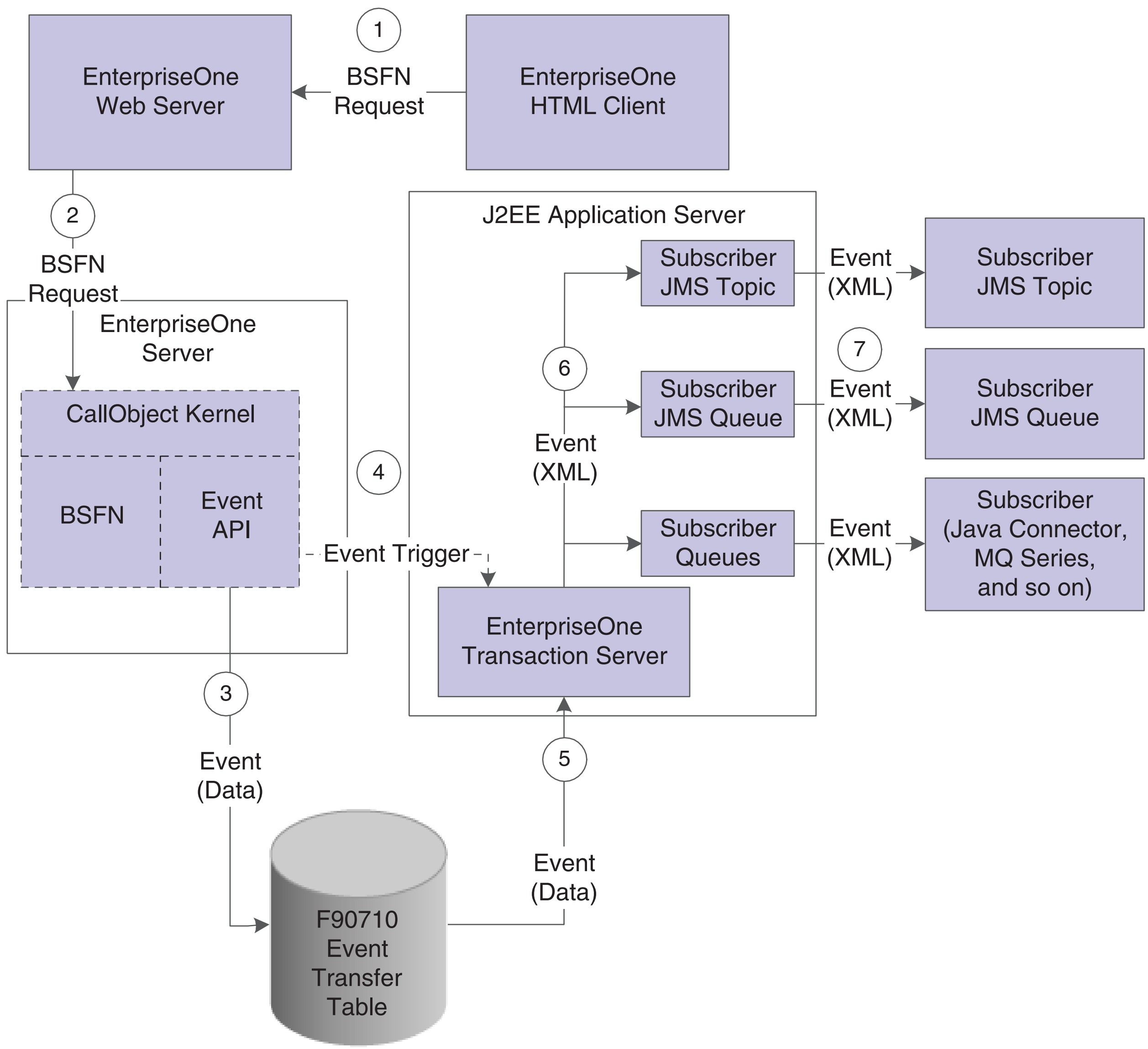
This section provides an overview of the architecture for processing events and discusses:

- Aggregating events
- Logging events
- Configuring the transaction server

14.2.1 Understanding Guaranteed Events Processing

This diagram provides an overview of the JD Edwards EnterpriseOne events architecture:

Figure 14–1 Guaranteed Events architecture overview



In summary, this is the general sequence that happens for an event to be published:

1. An HTML client user executes a business function request that is sent to the JD Edwards EnterpriseOne Web server.
2. The request is forwarded to a CallObject kernel on the JD Edwards EnterpriseOne server.
3. The CallObject kernel executes the business function, which calls the Event API to send the event data to the F90710 table.

   If the event is a Z event, the data sent to the F90710 table is in its final XML format.
4. A trigger message is sent to the JD Edwards EnterpriseOne Transaction server that indicates that a new event is in the F90710 table.
5. The transaction server retrieves the event data from the F90710 table and, for real-time and XAPI events, converts the event data to an XML document in the appropriate format.

6. The transaction server routes the event to the subscriber queues and subscriber topics for each subscriber that has established an active subscription for that event.
7. When a subscriber connects to the transaction server, the subscriber receives all the events that exist in its subscription queue and subscription topic at that time.

> **Note:** XAPI and Z events require additional information for event processing, which is discussed in the respective XAPI and Z event chapters.

14.2.2 Aggregating Events

Events are classified as either a single event or a container event. A single event can contain a single data structure. A container event can contain one or more single events or one or more data structures. You cannot define a container event using both single events and data structures for that specific container event. For example, RTSOHDR and RTSODTL are usually defined as single real-time events that represent the data structures in the header and detail areas of a sales order. RTSOOUT is usually defined as a container real-time event that contains both RTSOHDR and RTSODTL.

14.2.3 Logging Events

Real-time and XAPI events do not exist in their XML form until they are processed by the transaction server. Therefore, it is not possible to log the XML event on the JD Edwards EnterpriseOne server. However, if debugging is selected, the debug log file for the CallObject kernel that generates the event displays `jdeIEO_EventFinalize called for XX`, where XX is an integer that represents the number of times that jdeIEO_EventFinalize has been called in that kernel.

If you select debug logging for the transaction server, the transaction server debug log file displays this message `Sending event:` followed by the event data, including the full XML content of the event when the transaction server processes an event. There is one of these messages for every active subscriber that has an active subscription to the event.

> **Caution:** When logging is selected for the Transaction server, be sure to remove global read access rights for the logging directory to ensure data privacy.

If you use the dynamic Java connector graphical subscription application, you have the capability of sending the XML content of all received events to a specified directory.

See "Understanding Java Connector Events" in the *JD Edwards EnterpriseOne Tools Connectors Guide*.

14.2.4 Configuring the Transaction Server

The transaction server uses Java Message Service (JMS) queues and topics to guarantee event delivery. When an event occurs in JD Edwards EnterpriseOne, the transaction server retrieves the event information and routes the information to subscriber JMS queues and topics for each subscriber that has established an active subscription for that event.

When you use JD Edwards EnterpriseOne Tools 8.94 with JD Edwards EnterpriseOne Applications 8.11, you must configure these settings in the jde.ini file on your enterprise server so that the transaction server can find the event system:

- RunningHost
- RunningPort

Note: When you install the transaction server components, host and port information is written to a readme.txt file that is typically located at `c:\Program Files\JD Edwards\E1TranSvr\EventProcessor` on the transaction server.

See *JD Edwards EnterpriseOne Tools Server Manager Guide* on My Oracle Support.

See *JD Edwards EnterpriseOne Tools Transaction Server Reference Guide* on My Oracle Support.

When you use JD Edwards EnterpriseOne Tools 8.95 or a later Tools release with JD Edwards EnterpriseOne Applications 8.10, 8.11, or a later JD Edwards EnterpriseOne Applications release, you must configure the Object Configuration Manager (OCM) so that the transaction server can find the event system. You must access OCM from the Interoperability Event Definition program (P90701A).

Note: The ptf.log file contains Transaction server version information. The ptf.log file is located in EventProcessor_WAR.war and JDENETServer_WAR.war.

14.2.5 Configuring the Transaction Server to Use WebLogic

You can process real-time events using a JMS queue that is located in a WebLogic server.

Important: Setting the WebLogic client jar (wlfullclient.jar) is required when the transaction server is hosted in an Oracle Application Server or a WebSphere Application Server and the JMS queue is located in the WebLogic server instance.

14.2.5.1 Setting the WebLogic Client Jar in an Oracle Application Server

If your transaction server is hosted in an Oracle application Server, copy the wlfullclient.jar to the following location:

```
OAS_INSTALL\profilesj2ee\TS_CONTAINER\applications\OSBTransferAgent_EAR.ear\lib
```

14.2.5.2 Setting the WebLogic Client Jar in a WebSphere Application Server

If your transaction server is hosted in a WebSphere Application Server, copy the wlfullclient.jar to the following location:

```
WAS_INSTALL\profiles\TS_PROFILES\InstalledApps\DENITWSW60Node2Cell\
OSBTransferAgent_EAR.ear\lib
```

14.3 Setting Up OCM for Guaranteed Events

This section provides an overview of setting up OCM for guaranteed events and discusses how to set up OCM.

14.3.1 Understanding OCM Setup for Guaranteed Event Delivery

When you use JD Edwards EnterpriseOne Tools 8.95 and later releases with JD Edwards EnterpriseOne Applications 8.10 and later releases, you must define the transaction server and transaction server port settings in OCM so that the transaction server can find the event system. You access OCM from the Interoperability Event Definition program (P90701A). Once you access OCM from the Interoperability Event Definition program, you select the appropriate machine name and data source combination. This information should already be set up. If it is not, check with your System Administrator or refer to the Configurable Network Computing Implementation Guide for information about setting up OCM.

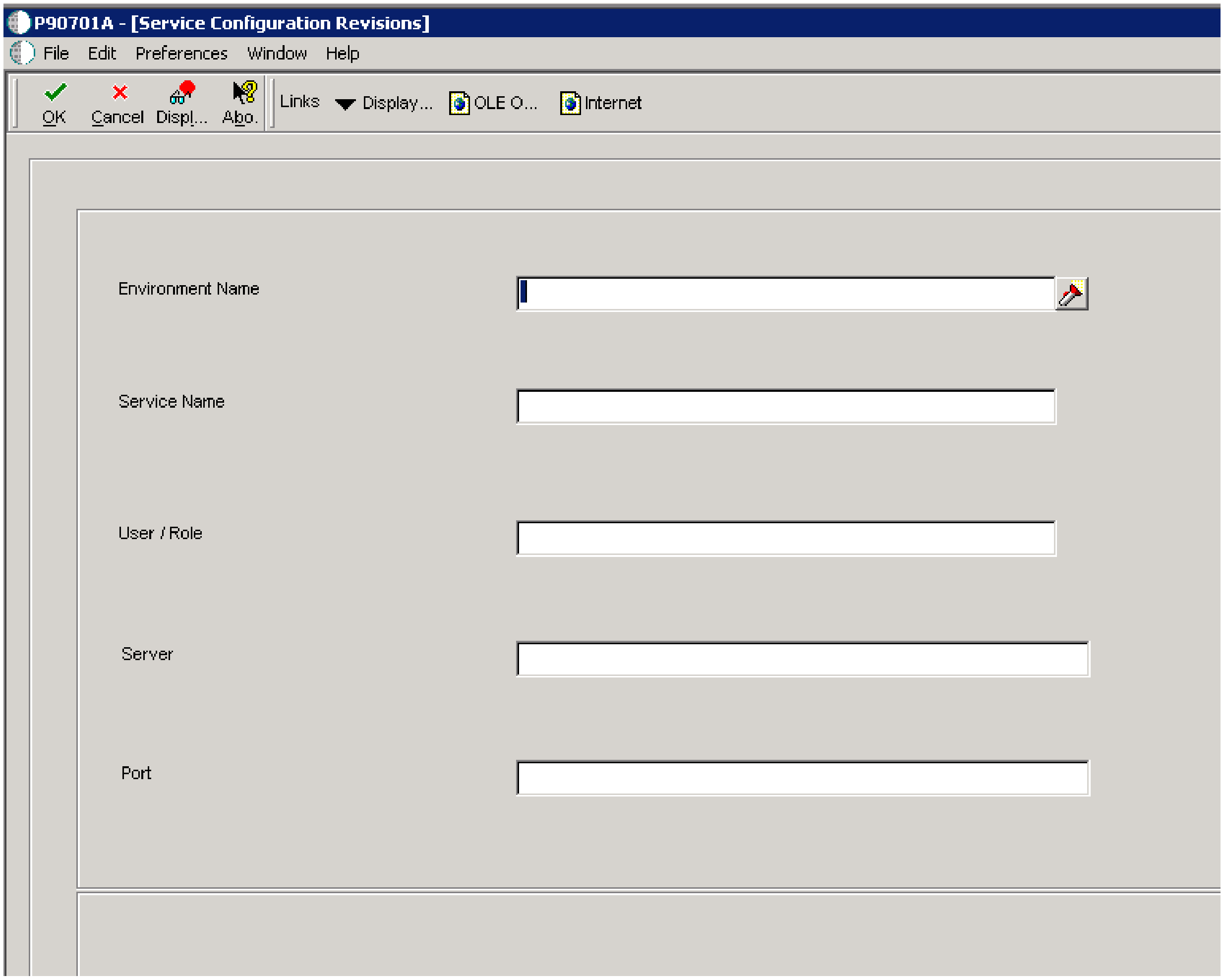
14.3.2 Forms Used to Set Up OCM for Guaranteed Event Delivery

Form Name	FormID	Navigation	Usage
Event Definition Workbench	W90701AA	Enter **P90701A** in the Fast Path Command Line.	Configure the OCM so the transaction server can find the event system.
Machine Search and Select	W986110D	From the Form menu on Event Definition Workbench, select Configure Servers.	Select the appropriate machine name and data source combination.
Work with Service Configurations	W986110J	On Machine Search and Select, select the machine name and data source combination and then click Select.	Find and select an existing configuration for the transaction server and server port or to access the Work with Service Configurations form to add a new configuration record for your transaction server.
Service Configuration Revisions	W986110K	On Work with Service Configurations, click Add.	Configure the OCM with the J2EE Transaction server and port.

14.3.3 Setting Up the OCM for Guaranteed Event Delivery

Access the Service Configuration Revisions form.

Figure 14–2 Service Configuration Revisions



Environment Name
A name that uniquely identifies the environment.

Service Name
A name that identifies the type of server. For example, RTE identifies the Transaction server.

User / Role
A profile that classifies users into groups for system security purposes. Use group profiles to give the members of a group access to specific programs.

Server
The name of the Transaction server.

Port
The port number of the Transaction server. This is the JDENET listening port.

14.4 Selecting the Guaranteed Events Delivery System

This section provides an overview of selecting the Guaranteed Events Delivery system and discusses how to select guaranteed events delivery.

14.4.1 Understanding Guaranteed Events Selection

JD Edwards EnterpriseOne Tools software is delivered with this functionality deselected. You perform this task only if you use JD Edwards EnterpriseOne Tools 8.95 with JD Edwards EnterpriseOne Tools 8.10, and you want guaranteed event delivery. Typically this task is performed by a system administrator. Use the Activate/Deactivate Guaranteed Event Delivery program (P90701A) to select or deselect the Guaranteed Events Delivery system.

Caution: Perform this task *only* if you use JD Edwards EnterpriseOne Tools 8.95 *with* JD Edwards EnterpriseOne Applications 8.10, *and* you want to use the Guaranteed Event Delivery system.

When you use JD Edwards EnterpriseOne Tools 8.95 and later tools releases with JD Edwards EnterpriseOne Applications 8.11 and later applications releases, the Guaranteed Event Delivery system is automatically available, and you do not perform this task. You are ready to define your events.

If you use JD Edwards EnterpriseOne Tools 8.95 with JD Edwards EnterpriseOne Tools 8.10 and do not perform this task, your events will be generated using the Classic Event Delivery System.

See Understanding Classic Events.

14.4.2 Forms Used to Select Guaranteed Events Delivery System

Form Name	FormID	Navigation	Usage
Event Definition Workbench	W90701AA	Type P90701A on the Fast Path.	Locate and review existing single and container events.
Activate Guaranteed Delivery	W90701AK	On Event Definition Workbench, select Guaranteed Events from the Form menu.	To select or deselect Guaranteed Events Delivery.

14.4.3 Selecting Guaranteed Events Delivery

Access the Activate Guaranteed Delivery form.

Activate Guaranteed Delivery
An option that enables you to select the Guaranteed Event Delivery system when you use JD Edwards EnterpriseOne Tools 8.95 with JD Edwards EnterpriseOne Applications 8.10.

14.5 Defining Events

This section provides an overview of defining events in the F90701 table and discusses how to add single and container events.

14.5.1 Understanding Events Definition

You use the Interoperability Event Definition program (P90701A) to define each real-time and XAPI event in JD Edwards EnterpriseOne. You use a separate process to define Z events, which is documented in the *Guaranteed Z Events* chapter.

Every real-time or XAPI event that you use in your system must have an associated record in the F90705 table. The F90705 table enables each event to be activated or deactivated for each environment in your system. When you create a new event, select the Create Activation Record option. When you add a new environment to your system, you must run the Populate Event Activation Status Table UBE (R90705) to create event activation records for existing events. The Populate Event Activation Status Table UBE is described in the JD Edwards EnterpriseOne Server Installation Guide.

After you define a new event, you must refresh the cache of active events on the transaction server. You can refresh the active events cache while the transaction server is running. If the transaction server is not running when this operation is performed, it automatically refreshes its cache when it is brought back to operational status.

See Also:

- Understanding Z Events - Guaranteed.

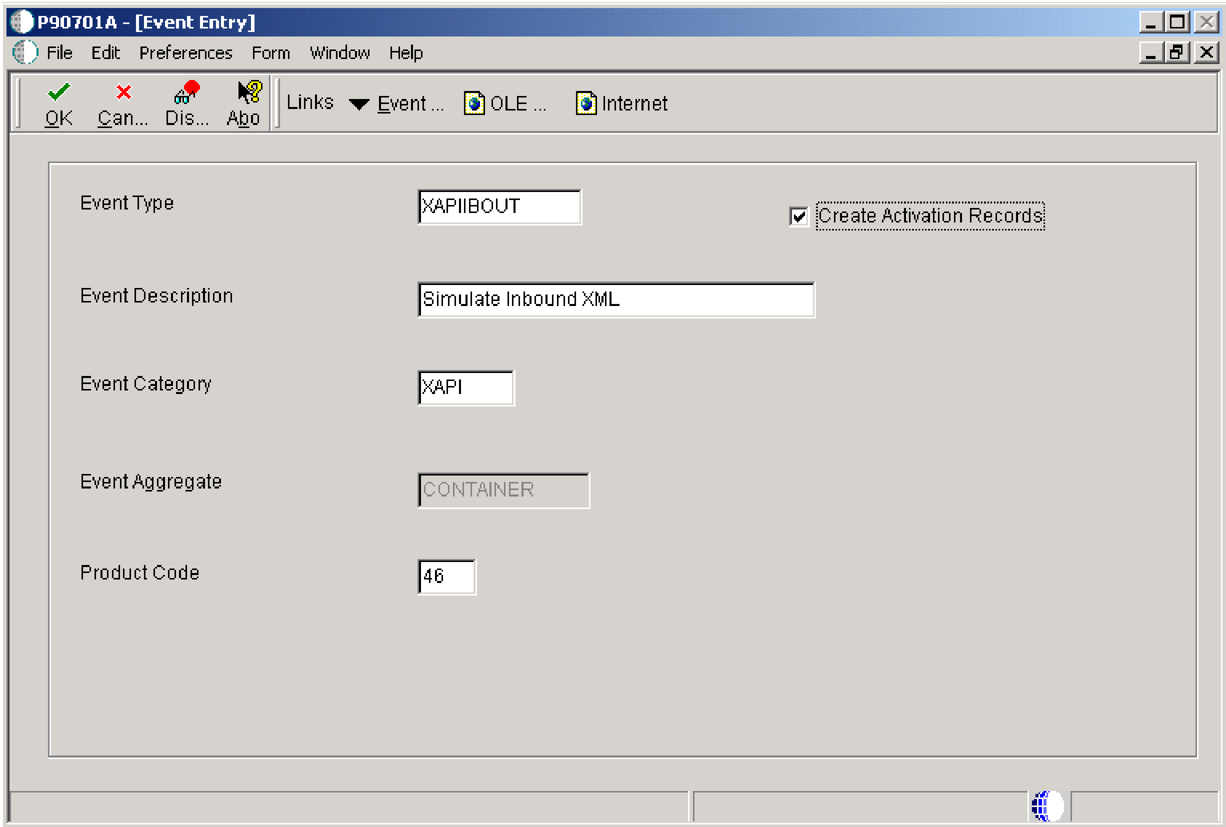
14.5.2 Forms Used to Enter Events

Form Name	FormID	Navigation	Usage
Event Definition Workbench	W90701AA	Type P90701A on the Fast Path.	Locate and review existing single and container events.
Event Entry	W90701AD	On Event Definition Workbench, click Add.	Add or change a single or container event.
Event Definition Detail	W90701AC	Automatically appears when you click OK on the Event Entry form if you entered **Container** in the Event Category field for a real-time event or if you entered **XAPI** in the Event Type field.	Link single events or data structures to a container event.
Event Activation by Environment	W90701AG	On Event Definition Workbench, select Event Activation from the Form menu.	Locate and review existing environments and event types.
Add Event Activation by Environment	W90701AH	On Event Activation by Environment, click Add.	To activate an event on a specific environment.

14.5.3 Adding a Single or Container Event

Access the Event Entry form.

Figure 14–3 Event Entry form



Event Type
The name of the event (for example RTSOOUT, which is the usual event type for a real-time sales order event).

Create Activation Records
An option that causes newly defined events to have an associated record in the F90705 table, which enables each event to be activated or deactivated for each environment in your system. You must select this option for every event that you intend to use in your system.

Event Description
The description of an event.

Event Category
A value that represents the name of the event type. Use RTE for real-time events or XAPI for XAPI events.

Event Aggregate
Indicates whether an event is a single event or a container event.

Product Code
An optional field that indicates to which JD Edwards EnterpriseOne system the event is associated.

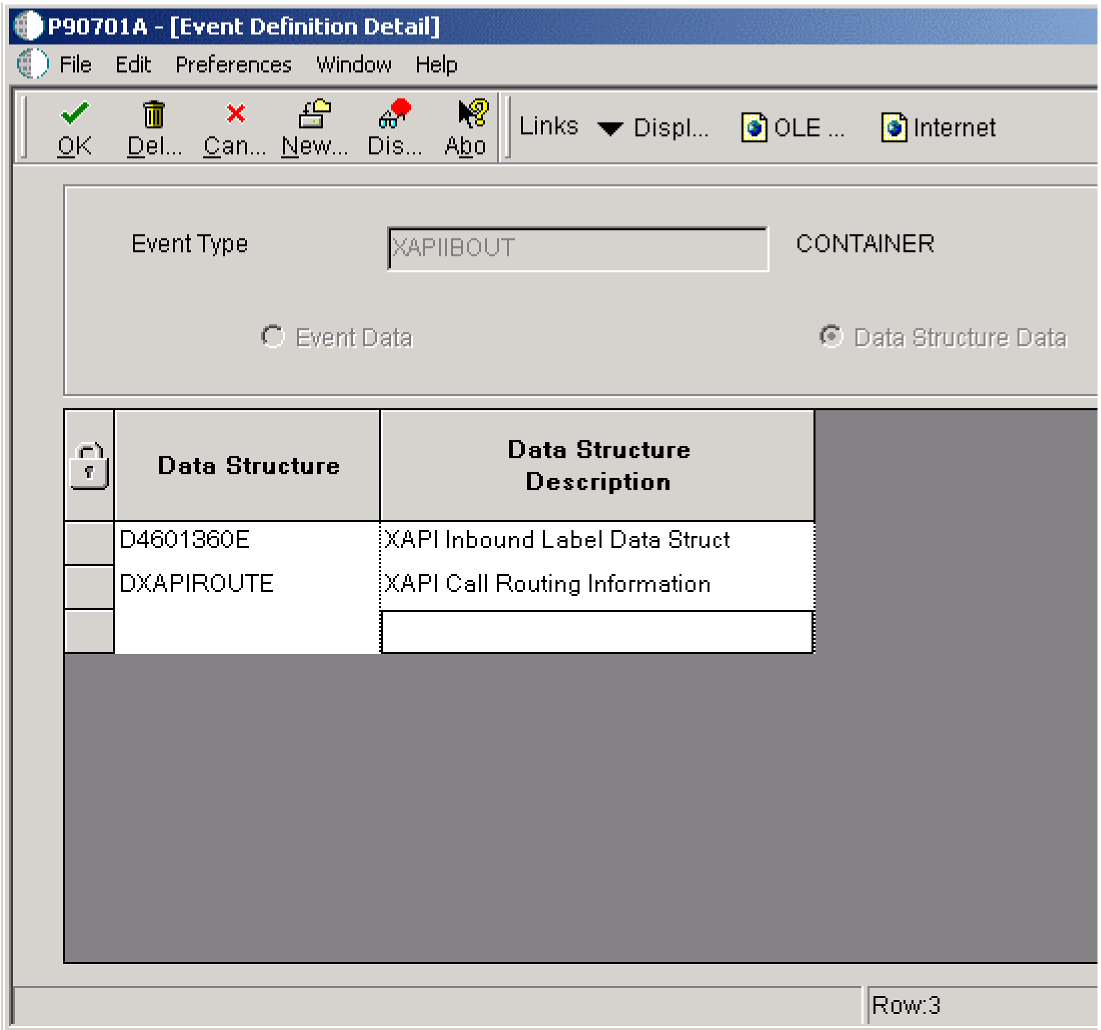
Data Structure
The name of the data structure that passes event information.

This field disappears if **Container** is the value of the Event Aggregate field; however, when you click OK, the Event Definition Detail form automatically appears for you to enter data structure information.

14.5.3.1 Event Definition Detail

Access the Event Definition Detail form.

Figure 14–4 Event Definition Detail form



Event Data
An option that enables you to define single individual events for a container event.

Data Structure Data
An option that enables you to define aggregate events for the container event. For XAPI events, you must select the Data Structure Data option.

14.5.3.2 Activating an Event

Access the Add Event Activation by Environment form

Environment

Your operating environment, such as Windows 2000, Windows NT, UNIX, IBM i, and so on.

14.5.3.3 Refreshing the Transaction server cache of active events

Access the Event Definition Workbench form.

To refresh the cache of active events with the Transaction server running, select Refresh Event Cache from the Form menu.

14.6 Establishing Subscriber and Subscription Information

This section provides an overview of subscriber and subscription information and discusses how to:

- Set up processing options for adding JMS Queue as a subscriber.
- Add a subscriber.
- Add a subscription.
- Associate a subscription with subscribed events.
- Associate a subscription with subscribed environments.

14.6.1 Understanding Subscribers and Subscriptions

You use the Interoperability Event Subscription program (P90702A) to establish subscribers and to add subscriptions. After you add a subscriber, you must activate it. If your subscriber is inactive, you will not receive any events even if you have active subscriptions. You activate subscribers on the Event Subscribers form by selecting the subscriber, and then selecting Change Status from the Row menu.

Each subscriber can have one or more subscriptions. Each subscription can be associated with one or more subscribed events and subscribed environments. Each subscription that you want to use must be activated. You activate subscriptions on the Event Subscriptions form by selecting the subscription, and then selecting Change Status from the Row menu.

Any time you make a change to a subscriber, including the associated subscriptions, you must refresh the subscriber cache on the JD Edwards EnterpriseOne and the Transaction servers for the changes to become effective. You can refresh your running system from the Event Subscribers form by selecting Refresh Sub Cache from the Form menu.

Oracle's Enterprise Service Bus (ESB) is a subscriber that uses the JMS Queue and JMS Topics transports. Oracle Service bus (OSB) is a subscriber that uses the JMS Queue transport. You can set up processing options for EDB (WebSphere) and OSB (WebLogic) so that when you add JMS Queue as a new subscriber, the value for the Initial Context Factory and provider URL fields are entered by the system.

14.6.2 Forms Used to Add a Subscriber and Subscription Information

Form Name	FormID	Navigation	Usage
Event Subscribers	W90702AA	Type P90702A in the Fast Path.	Locate and review existing subscribers.

Form Name	FormID	Navigation	Usage
Add Event Subscriber	W90702AB	On Event Subscribers, click Add.	Add or change a subscriber.
Event Subscriptions	W90702AD	Select a subscriber in the detail area of the Event Subscribers form, and then select Event Subscriptions from the Row menu.	Locate and review existing subscriptions for a subscriber.
Add Event Subscription	W90702AE	On Event Subscriptions, click Add.	Add new subscription information.
Subscribed Events	W90702AG	On the Event Subscriptions form, select the subscription information in the detail area, and then select Subscribed Events from the Row menu.	Associate a subscription with an event.
Subscribed Environments	W90702AF	On the Event Subscriptions form, select the subscription information in the detail area, and then select Subscribed Env from the Row menu.	Associate a subscription with an environment.

14.6.3 Setting Up Processing Options for Adding JMS Queue as a Subscriber

Access the Interactive Versions form on JD Edwards EnterpriseOne by typing IV in the Fast path. Use these processing options to define values for adding JMS Queue as a subscriber.

Oracle Application Server Initial Context Factory

Use this processing option to specify the value for the Initial Context Factory field that appears on the Add Event Subscriber form. The value you enter in this processing option appears in the Add Event Subscriber form when the Application Server is defined as **Oracle**. The default value is *com.evermind.server.rmi.RMIInitialContextFactory.*

Oracle Application Server Local Provider URL

Use this processing option to specify the value for the Provider URL field that appears on the Add Event Subscriber form. The value you enter in this processing option appears in the Add Event Subscriber form when the Application Server is defined as **Oracle** and the Queue Location is defined as **Local.** The default value is *ormi://localhost:23791.*

Oracle Application Server Remote Provider URL

Use this processing option to specify the value for the Provider URL field that appears on the Add Event Subscriber form. The value you enter in this processing option appears in the Add Event Subscriber form when the Application Server is defined as **Oracle** and the Queue Location is defined as **Remote**. The default value is *ormi://remote-machine-name:23791.*

WebSphere Initial Context Factory

Use this processing option to specify the value for the Initial Context Factory field that appears on the Add Event Subscriber form. The value you enter in this processing option appears in the Add Event Subscriber form when the Application Server is defined as **WebSphere.** The default value is *com.ibm.websphere.naming.WsnInitialContextFactory.*

WebSphere Local Provider URL

Use this processing option to specify the value for the Provider URL field that appears on the Add Event Subscriber form. The value you enter in this processing option appears in the Add Event Subscriber form when the Application Server is defined as **WebSphere** and the Queue Location is defined as **Local.** The default value is *corbaloc:iiop:localhost:2809.*

WebSphere Remote Provider URL

Use this processing option to specify the value for the Provider URL field that appears on the Add Event Subscriber form. The value you enter in this processing option appears in the Add Event Subscriber form when the Application Server is defined as **WebSphere** and the Queue Location is defined as **Remote.** The default value is *corbaloc:remote-machine-name:2809.*

WebLogic Initial Context Factory

Use this processing option to specify the value for the Initial Context Factory field that appears on the Add Event Subscriber form. The value you enter in this processing option appears in the Add Event Subscriber form when the Application Server is defined as **WebLogic**. The default value is *weblogic.jndi.WLInitialContextFactory*.

WebLogic Local Provider URL

Use this processing option to specify the value for the Provider URL field that appears on the Add Event Subscriber form. The value you enter in this processing option appears in the Add Event Subscriber form when the Application Server is defined as **WebLogic** and the Queue Location is defined as **Local**. the default value is *t3://localhost:7001*.

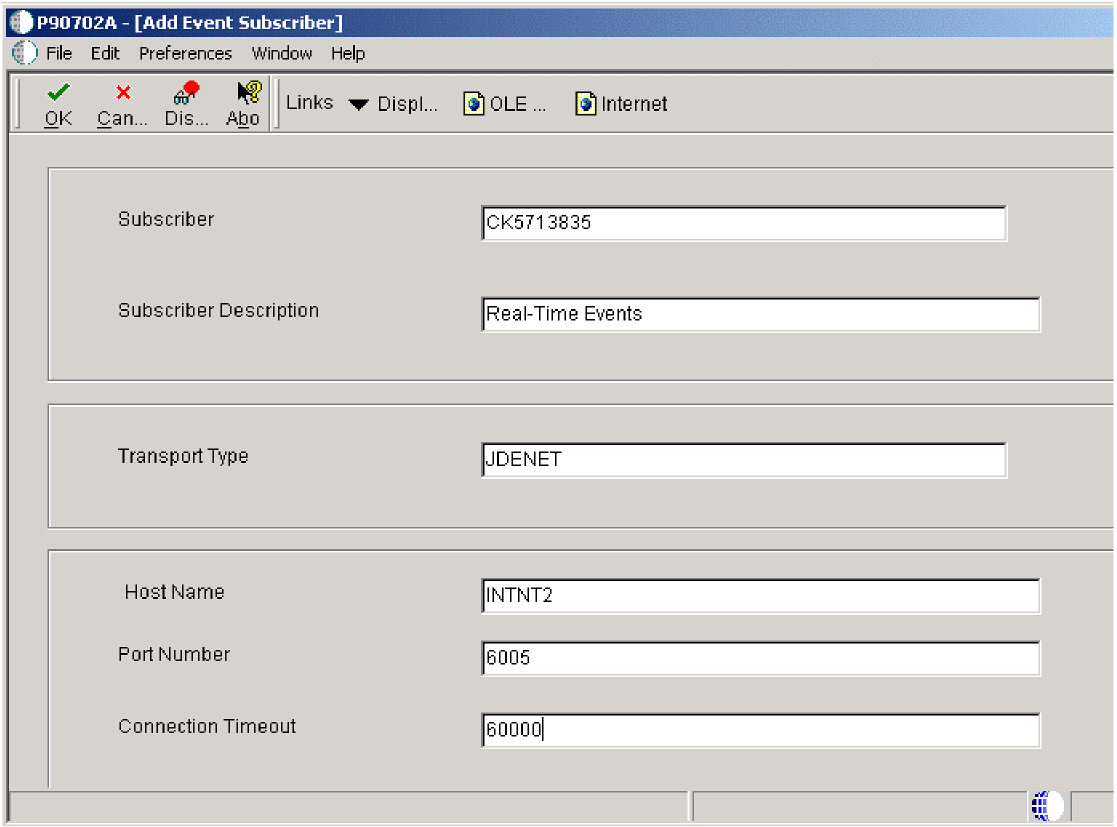
WebLogic Remote Provider URL

Use this processing option to specify the value for the Provider URL field that appears on the Add Event Subscriber form. The value you enter in this processing option appears in the Add Event Subscriber form when the Application Server is defined as **WebLogic** and the Queue Location is defined as **Remote**. The default value is *t3://remote-machine-name:7001*.

14.6.4 Adding a Subscriber

Access the Add Event Subscriber form.

Figure 14–5 Add Event Subscriber form



Subscriber
The JD Edwards EnterpriseOne user ID for the user who is to receive the subscribed events.

Subscriber Description
A description of the subscriber.

Transport Type
Describes through which mechanism the subscriber receives events. Valid transport types are:

- COMCONN: COM Connector
- JAVACONN: Java Connector (including WSG)
- JDENET: For XAPI requests

  Additional fields appear on the Add Event Subscriber form. In the Host Name field, enter the name of the server that processes events for the subscriber. In the Port Number field, enter the port where the subscriber service is running. In the Connection Timeout field, enter the time in milliseconds after which the event connection is considered timed out.
- JMSTOPIC: JMS Topic

  Additional fields appear on the Add Event Subscriber form. In the Connection Factory JNDI field, enter the JMS Topic Connection Factory JNDI name. In the Topic Name field, enter the JMS Topic name for your subscriber.

Important: The values that you enter in the Connection Factory JNDI Name field and the Topic Name field must be the same values that you configured on the WebSphere Application Server

---

See *JD Edwards EnterpriseOne Tools Transaction Server Reference Guide* on My Oracle Support.

- JMSQUEUE: JMS Queue

  These additional fields appear on the Add Subscriber Event form. In the Connection Factory JNDI field, select the JMS Queue Connection Factory JNDI name from the drop-down list. In the Queue Name field, select **JMSQUEUE** from the drop-down list. Verify the value in the Message Format field is correct. Verify the value in the Application Server field is correct. You can search and select one among the three application servers (Oracle Application Server, WebLogic, and WebSphere Application Server). This server entry affects the value that the system enters in the Initial Context Factory field and the Provider URL field. In the Queue Location field, select the appropriate value from the drop-down list. User Local if the queue and the transaction server are on the same application server. Use Remote if the queue and the transaction server are on different applications servers. After you enter the value in the Queue Location field, the system updates the Initial Context Factory field and Provider URL field. You can change these values.

  Important: The value that you enter in the Connection Factory JNDI Name field must be the same value that you configured on the Oracle Application Server, WebSphere Application Server, or WebLogic Application Server.

  See *JD Edwards EnterpriseOne Tools Transaction Server Reference Guide* on My Oracle Support.

- MQSQ: IBM WebSphere MQ

  Additional fields appear on the Add Subscriber Event form. In the Connection Factory JNDI field, enter the WebSphere MQ Connection Factory JNDI name. In the Queue Name field, enter the WebSphere MQ queue name for your subscriber

  See Creating WebSphere MQ Queues.

- MSMQ: Microsoft Message Queue

  Additional fields appear on the Add Subscriber Event form. In the Queue Label field, enter the MSMQ Queue Label. In the Queue Name field, enter the MSMQ Queue Name

14.6.5 Adding a Subscription

Access the Add Event Subscription form.

Subscriber
The JD Edwards EnterpriseOne user ID for the user who is to receive the subscribed events.

Subscription Name
A unique name for the subscription.

Subscription Description
A description of the subscription.

14.6.6 Associating a Subscription with Subscribed Events

Access the Subscribed Events form.

Event Type
The name of the event.

14.6.7 Associating a Subscription with Subscribed Environments

Access the Subscribed Environments form.

Environment
The JD Edwards EnterpriseOne environment with which the subscription is associated. Each subscription can be associated with any number of valid environments.

14.7 Creating MSMQ Queues

This section provides an overview about MSMQ and discusses how to:

- Create an MSMQ real-time event queue.
- Verify event delivery.

14.7.1 Prerequisites

Before you complete this task:

- MSMQ is installed on your system.
- WebSphere is installed on your system.

14.7.2 Understanding MSMQ

You can use Microsoft message queueing to subscribe to and receive events. After you create the events queue for MSMQ, you must add the queue name as a subscriber, using the Interoperability Event Subscription program (P90702A). The queue name must be in MSMQ direct format, which includes your machine name or IP address, depending on which protocol you use. Naming conventions for MSMQ direct format queue names are discussed on Microsoft's web page.

After you create the queue and set up the subscriber information, you should verify event delivery. MSMQ RTEII, a server-only feature, is an extension of COMConnector.

14.7.3 Creating an MSMQ Real-Time Event Queue

Use these steps to configure MSMQ:

1. From the Control Panel, select Administrative Tools, and then select Computer Manage.
2. On the Computer Management Console, navigate to Services and Applications, and then open Message Queuing.
3. Open Private Queue, right-click the Private Queue folder, select New, and then Private Queue.

Note: You can create the events queue under Public Queue if you prefer.

---

4. In Queue Name, select a meaningful queue name, for example, RTE-TEST.
5. If the events queue is used in a transactional environment, select the Transactional option, and then click OK.

Note: If you are creating an event queue in a transactional environment, you must use a private (remote) queue.

---

6. Right-click your newly created events queue and select properties.
7. In the Label field, enter a meaningful queue label name; for example, E1Outbound, and then click OK.

14.7.4 Verifying Event Delivery

Use these steps to verify event delivery:

1. Start your COMConnector on your enterprise server.

Note: Do not start your COMConnector on your client.

---

2. On your enterprise server, in MSMQ Computer Management, select the queue that you configured to receive JD Edwards EnterpriseOne events.
3. To see if any events are in the queue, click the queue messages under queue name and select Action then Refresh in the Computer Management menu.
4. Double-click any messages that are in the queue.

   A menu displays the message content up to the first 256 bytes.

14.8 Creating WebSphere MQ Queues

This section provides an overview about WebSphere MQ and discusses:

- Creating a WebSphere MQ real-time event queue.
- Configuring WebSphere.
- Verifying event delivery.

14.8.1 Prerequisites

Before you complete this task:

- WebSphere MQ is installed on your system with PTF CSD06.
- WebSphere is installed on your system.

14.8.2 Understanding WebSphere MQ

You can use IBM's message queueing to subscribe to and receive events. After you create the events queue for WebSphere MQ, you must add the queue name as a subscriber, using the Interoperability Event Subscription program (P90702A).

After you create the queue and set up the subscriber information, you should verify event delivery.

14.8.3 Creating a WebSphere MQ Real-Time Event Queue

Use these steps to configure WebSphere MQ:

1. Open the WebSphere MQ Explorer and navigate to the Queue Manager.

   The default queue manager is typically named QM_<hostname>, where <hostname> is the machine name where WebSphere MQ is installed.

   > **Note:** If the QM_<hostname> queue is not created, then manually create the queue. Right-click Queue Managers, select New, and then select Queue Manager. Complete the data fields on each successive screen.

2. Under Queue Manager, select the Queues folder.

   This shows any existing queues hosted by this queue manager.

3. To create the queue for delivery of JD Edwards EnterpriseOne events, select New then Local Queue from the Action menu on the WebSphere MQ Explorer.

   > **Note:** On Create Local Queue, enter a meaningful queue name, for example, RTE_TEST_QUEUE.

4. To make the queue persistent, select the Persistent option for the Default Persistence field.

   The default settings should be sufficient for the remaining configuration values.

   > **Important:** When entering queue names for IBM WebSphere MQ, the queue name *must* be all upper case.

14.8.4 Configuring WebSphere

Use these steps to configure WebSphere:

1. Log on to the WebSphere Administration Console.
2. Create a Queue Connection Factory by selecting WebSphere MQ JMS Provider under Resources.

   Enter a meaningful connection factory name along with a JNDI name; for example, jms/mq/rte/QueueConnectionFactory.

   > **Note:** When you add a WebSphere MQ subscriber in JD Edwards EnterpriseOne, enter this name in the Connection Factory JNDI field.

3. Create a queue destination by selecting WebSphere MQ JMS Provider under Resources.
   a. In the Name and Base Queue Name field, enter the same queue name that you used when you created the queue in the WebSphere MQ Explorer; for example RTE_TEST_QUEUE.
   b. Enter a meaningful JNDI name; for example, jms/mq/rte/TestQueue01.

      Note: When you add a WebSphere MQ subscriber in JD Edwards EnterpriseOne, enter this name in the Queue Name field.

   c. Enter the Queue Manager name; for example, QM_DENNF13.
4. Save these changes in the WebSphere console.

14.8.5 Verifying Event Delivery

Use these steps to verify event delivery:

1. In the WebSphere MQ Explorer, select the queue you configured to receive JD Edwards EnterpriseOne events.

   Note: To see if any events are in the queue, click the refresh button on the Explorer window. The Current Depth column shows the number of messages in the queue. You might have to scroll right in the explorer window to see this column.

2. If there are messages in the queue, right-click the queue.
3. To see the messages in the queue, select Browse Messages in the pop-up menu.

   Note: JD Edwards EnterpriseOne sends the event XML to an WebSphere MQ queue, not the serialized object sent to subscriber queues serviced by the Java connector.

14.9 Creating WebLogic Message Queues

This section provides an overview about WelbLogic messaging queues and discusses:

- Creating a JMS server in the WebLogic server.
- Creating a JMS module in the WebLogic server.
- Creating a connection factory.
- Creating a destination (queue).
- Verifying event delivery.

14.9.1 Prerequisites

Before you complete this task:

- WebLogic is installed on your system.
- The transaction server is configured for your application server.

See Configuring the Transaction Server to Use WebLogic.

14.9.2 Understanding WebLogic Message Queue

You can use Oracle Business Service (OSB) message queue to subscribe to and receive events. After you create the events queue for WebLogic, you must add the queue name as a subscriber, using the Interoperability Event Subscription program (P90702A).

After you create the queue and set up the subscriber information, you should verify event delivery.

14.9.3 Creating a JMS Server in the WebLogic Server

Use these steps to create a JMS server in the WebLogic server:

1. In the WebLogic admin console, go to Home > Summary of Services: JMS > Summary of JMS Servers.
2. Click Lock & Edit.
3. Click New.
4. On Create a New JMS Server, enter a name for your JMS server in the Name field.
5. Click the Create New Store button.
6. Click Next.

14.9.4 Creating a JMS Module in the WebLogic Server

Use these steps to create a JMS module in the WebLogic server:

1. In the WebLogic admin console, go to Home >Summary of Services: JMS > JMS Modules.
2. To create a new module, type the module name in the Name field.
3. Accept the default values for the Descriptor File Name and Location In Domain fields.
4. Click Next.

14.9.5 Creating a Connection Factory

Use these steps to create a Connection Factory in WebLogic:

1. From the WebLogic Admin Console, go to Home >JMS Modules > jmsModule.
2. To create a new connection factory, enter a meaningful connection factory name in the Connection Factory field.
3. Set the JNDI name to **OSBSubscriberQCF**.
4. Click Next.

14.9.6 Creating a Destination

Use these steps to create a destination (queue):

1. From the WebLogic Admin Console, go to Home > JMS Modules > jmsModule.
2. To create a new queue, enter the queue name in the Create a New Queue field.
3. Set the JNDI name to **OSBSubscriber** Queue.

4. Click Next.
5. Enter a meaningful name in the Subdeployments field.
6. Click Next.

14.9.7 Verifying Event Delivery

Use these steps to verify event delivery:

- In the WebLogic message queue, select the queue you configured to receive JD Edwards EnterpriseOne events.

> **Note:** To see if any events are in the queue, click the Refresh button on the Explorer window. The Current Depth column shows the number of messages in the queue. You might have to scroll right in the explorer window to see this column.

- If there are messages in the queue, right-click the queue.
- To see the messages in the queue, select Browse Messages in the pop-up menu.

> **Note:** JD Edwards EnterpriseOne sends the event XML to a WebLogic message queue, not the serialized object sent to subscriber queues serviced by the Java connector.

14.10 Creating Custom Real-Time Events

This section discusses how to create a real-time event.

14.10.1 Creating a Custom Real-Time Event

JD Edwards EnterpriseOne provides predefined real-time events that capture certain JD Edwards EnterpriseOne transactions and notify subscribers about the transaction. If you have requirements that are not satisfied by the predefined real-time events, you can create a custom real-time event. Chapter 14, Using Real-time Events – Guaranteed, of the Interoperability Guide provides conceptual information about real-time events, identifies APIs for creating real-time events, and provides sample code.

Before you create a custom real-time event, you should review the existing real-time events to determine if there is one that you can use as a model for creating your custom real-time event. Detail information about each real-time event can be found in the Application Real-Time Events Implementation Guide.

See *JD Edwards EnterpriseOne Application Real-Time Events Implementation Guide*.

Use the following steps to create a custom real-time event. Each step includes a reference to documentation that provides more information about that step.

1. Determine the type of real-time event (single, aggregate, or composite).

   See Understanding Real-Time Event Generation.

2. Create a new data structure or modify an existing data structure to pass data.

   See "Creating Data Structures" in the *JD Edwards EnterpriseOne Tools Development Tools: Data Structure Design Guide*.

3. Create a new event definition.

See Defining Events.

4. Create a new business function or modify an existing business function to call the API that generates the event.

   See Using Business Function Calls.

   See "Understanding Business Functions" in the *JD Edwards EnterpriseOne Tools Development Tools: APIs and Business Functions Guide*.

   See "Development Standards for Business Function Programming Overview" in the *JD Edwards EnterpriseOne Tools Development Standards for Business Function Programming Guide*.

   See *JD Edwards EnterpriseOne Tools API Reference Guide* on My Oracle Support.

5. Build and promote the business function.

   See "Understanding Package Management" in the *JD Edwards EnterpriseOne Tools Package Management Guide*.

6. Add the subscriber, associate the event to the subscriber, and enable the subscription.

   See Establishing Subscriber and Subscription Information.

7. Configure Object Configuration Manager (OCM) for Guaranteed Event Delivery.

   See Setting Up OCM for Guaranteed Events.

8. Configure and start your servers (transaction, integration, and enterprise) and test the real-time event.

   See Understanding Guaranteed Events Processing.

   See *JD Edwards EnterpriseOne Tools Server Manager Guide* on My Oracle Support.

   See *JD Edwards EnterpriseOne Tools Reference Guide* on My Oracle Support.

   See *JD Edwards EnterpriseOne Tools Transaction Server Components Reference Guide* on My Oracle Support.

14.11 Generating Schemas for Event XML Documents

This section provides an overview of the Schema Generation Utility and discusses how to:

- Configure the Schema Generation Utility.
- Use the Schema Generation Utility.
- Troubleshoot the Schema Generation Utility.

14.11.1 Understanding the Schema Generation Utility

The Schema Generation Utility creates XML schemas from event definitions. The purpose of this utility is to facilitate orchestration developers who use orchestration systems such as Oracle's Enterprise Service BUS (ESB) or Business Process Execution Language Process Manager (BPEL-PM) to process real-time events, XAPI events, and Z events.

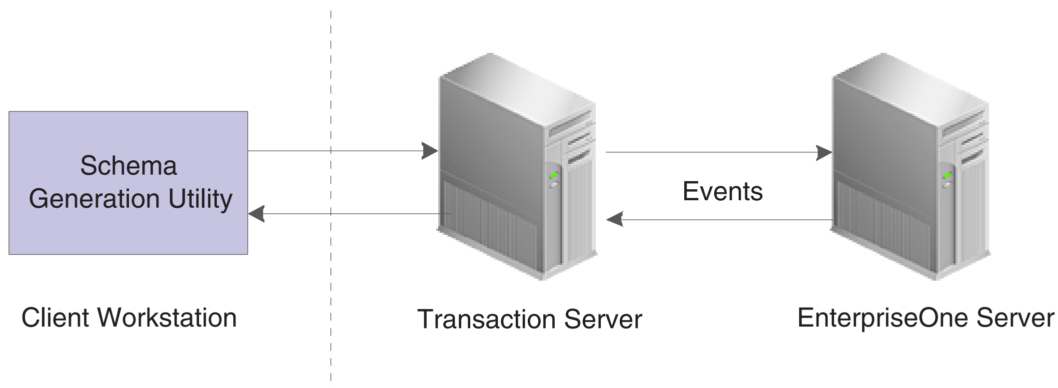
The Schema Generation Utility enables you to generate and save the schemas. The Schema Generation Utility generates schemas for these events:

- Single event: You can select a single event of a particular event category to generate schema.
- Multiple events: You can select multiple events of a particular event category to generate schemas.
- All events: You can generate schemas for all JD Edwards EnterpriseOne events of a particular event category.

In addition, the Schema Generation Utility can generate XML schema for a generic header representing all events. This schema can be used in orchestration systems for content-based routing.

This diagram provides an overview of the Schema Generation utility.

Figure 14–6 Schema Generation Utility



14.11.1.1 Prerequisite

Before you configure the Schema Generation Utility, you must install a Java Runtime Environment (JRE) version 1.4.0 or later on your local machine. You can download a JRE from the Sun Developer Network (SDN) web page (http://java.sun.com/).

14.11.2 Configuring the Schema Generation Utility

The Schema Generation Utility is delivered in a zip file in the system\classes folder. You must download the zip file to your local machine and set up certain files. You use these settings in Step 4.

Section	Setting	Value
[EVENTS]	initialContextFactory	For Oracle Application Server, the default value is: com.evermind.server.rmi.RMIInitialContextFactory For WebSphere Application Server, the default value is: com.ibm.websphere.naming.WsnInitialContextFactory

Section	Setting	Value
[EVENTS]	jndiProviderURL	For the Transaction Server running on an Oracle Application Server, the value is: jndiProviderURL=opmn:ormi://machine_ name:6003:e1transvr/EventProcessor For the Transaction Server running on a WebSphere Application Server, the value is: jndiProviderURL=corbaloc::Machine_ name:Port/NameServiceServerRoo **Note:** Machine_name in this setting is the name of the machine where the Transaction Server is installed Port in this setting is the Bootstrap Address port of the Transaction Server. Generally the port is 9810.
[EVENTS]	eventServiceURL	The value is: http://machine_ name:port/e1events/EventClientService **Note:** Machine_name in this setting is the name of the machine where the Transaction Server is installed, up, and running. Verify the hostport property in the jas.ini file of the Transaction Server for port information and to find the exact port for the URL.
[SECURITY]	SecurityServer	Provide the name of the user's EnterpriseOne Security Server.
[JDENET]	serviceNameConnect	Provide the port that you are connecting on to the user's EnterpriseOne Security Server.
[INTEROP]	enterpriseServer	Provide the name of the user's EnterpriseOne Server.
[INTEROP]	port	EnterpriseOne Server port.

To configure the Schema Generation Utility:

1. Navigate to the system\classes folder and unzip the SchemaGenUtil.zip file to the C:\SchemaGenUtil directory in your machine.

   Ensure to unzip the file with the full path information for each file in the zip file.

2. Configure the files in your C:\SchemaGenUtil\config directory.

   Ensure that the configured files have the .templ file extension removed from them. The proper filenames for that directory are jdbj.ini, jdeinterop.ini, and jdelog.properties.

3. Configure jdbj.ini and jdelog.properties files according to the environment.

   The simplest solution for the jdbj.ini file is to use the same file that has been configured on the Transaction Server.

   Note: See your JD Edwards EnterpriseOne systems administrator if you do not know the appropriate values for these files.

4. Configure the jdeinterop.ini file sections and settings that are identified in the preceding table.

5. Edit the C:\SchemaGenUtil\runSchemaGenUtilityDriver.bat file, pointing it to the location of the installed JRE.

14.11.3 Using the Schema Generation Utility

You use your JD Edwards EnterpriseOne user credentials to log into the Schema Generation Utility. Upon successfully logging in, the Event Schema Generator screen appears. This screen has two panels, Event Operations and Exception. You use the Event Operations panel to generate and display schemas for events. The Exception panel informs you of errors.

14.11.3.1 Prerequisites

Before you use the Event Schema Generator, ensure that:

- The event for which you want to generate a schema is active in the environment that you are using.
- The database driver file is in the classpath—if not, copy the database driver files to the following directory:

  C:\SchemaGenUtil\lib

14.11.3.2 Logging In to the Schema Generation Utility

To log in to the Schema Generation Utility:

1. On your local machine, navigate to the C:\SchemaGenUtil directory and double-click the runSchemaGenUtilityDriver.bat file.

   The Event Schema Generator sign-on window appears.

2. Enter your JD Edwards EnterpriseOne user credentials for these fields:

   – User Name

   – Password

   – Environment

     Events must be active in this environment.

   – Role

3. Select the Remember Sign On Info option if you want the system to remember your sign-on information, and then click OK.

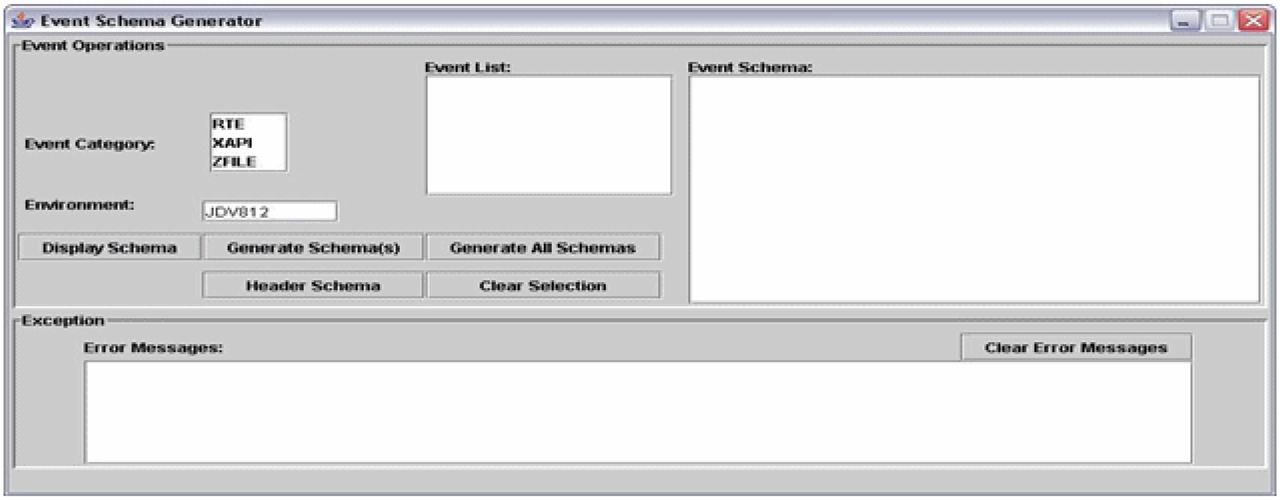
   Note: You can ignore the following warning message:

   Unable to initialize the management agent. Server Manager capability will be unavailable

14.11.3.3 Event Schema Generator Screen

After you successfully log in, the Event Schema Generator screen appears. This example shows the two panels on the Event Schema Generator screen:

Figure 14–7 Event Schema Generator



You use the Event Operations panel to provide information about the event or events for which you want schema generated. The utility provides the three event categories (real-time, XAPI, and Z events) from which you select and you must identify the environment. All events for which you want to generate a schema must be active in the environment that you indicate. When you select an event category and environment, the utility provides a list of events that are available.

You can perform the following tasks from the Event Operations panel:

- Display an event schema.
- Generate event schema for single and multiple events.
- Generate event schema for all the events of a selected event category.
- Generate header schema.

Click the Clear Selection button to clear the selection in the Event List panel. After the utility generates the schema, the schema is displayed in the Event Schema field.

If an error occurs during schema generation, the utility displays an error message in the Error Messages field of the Exception panel. You remove a message by clicking the Clear Error Message button.

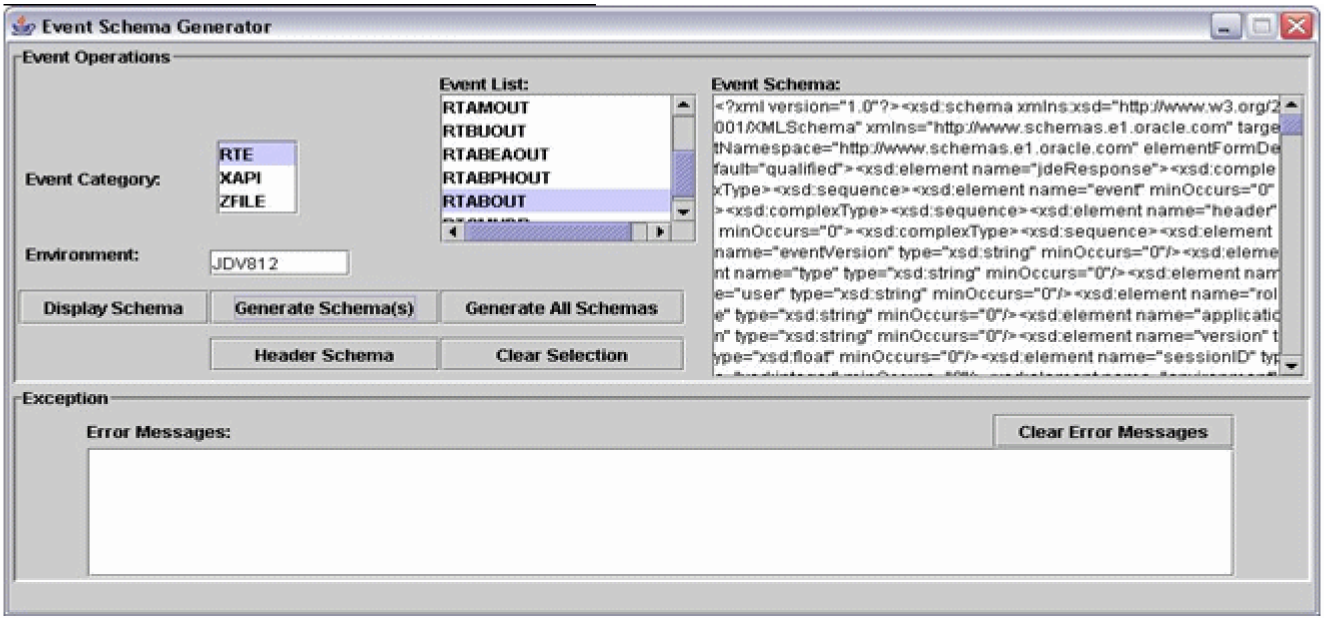
If no events are available in the given environment, the Schema Generation Utility displays an error message, such as *No event available for RTE in DEMOENV* in the Error Message panel.

You terminate the Schema Generation Utility by clicking Close at the right top of the main frame.

14.11.3.4 Displaying Event Schema

You can display event schema. This example shows how the utility displays a schema:

Figure 14–8 Event Schema Generator displaying event schema



To display event schema:

1. In the Event Operations panel of the Event Schema Generator screen, select the type of event from the Event Category field.
2. In the Environment field, enter the name of the environment that has the active event.
3. In the Event List field, select an event.

   Select only one event. If you select multiple events, the utility displays an error message in the Error Messages field of the Exception panel. The error message for selecting multiple events indicates invalid input. Click the Clear Selection button to clear a selection from the Event List.
4. Click Display Schema.

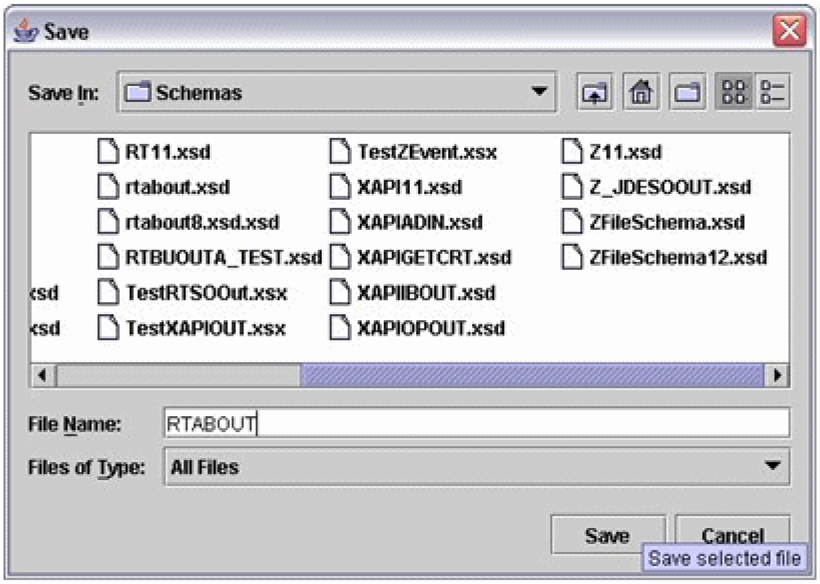
The utility displays the generated schema in the Event Schema field.

14.11.3.5 Generating Event Schema for Single and Multiple Events

You can generate schema for a single event and save the schema to a file. The utility saves generated schemas with a file extension of .xsd. After you complete the selection criteria and click Generate Schema(s) on the Event Schema Generator screen, a file chooser dialog screen appears. You indicate the file path and enter the file name.

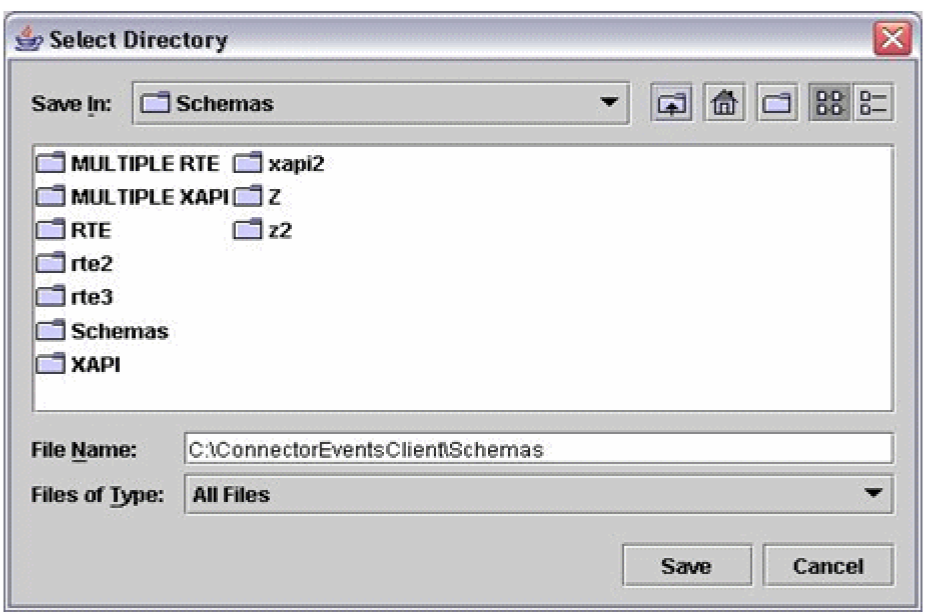
You can save the generated schema for a single event. This example is the file chooser dialog screen for saving a single event:

Figure 14–9 Save screen for a single event



You can generate schemas for multiple events and save the schemas to a directory. To select multiple events from the Event List field, press the Ctrl key and select the event. After you complete the selection criteria and click Generate Schema(s) on the Event Schema Generator screen, a Select Directory dialog screen appears. You enter the full path name where the directory is located. The utility saves each schema file as E1_EventType.xsd, for example, E1_RTSOOUT.xsd.

Figure 14–10 Select Directory screen for multiple events



To generate event schema for single and multiple events:

1. In the Event Operations panel of the Event Schema Generator screen, select the type of event from the Event Category field.
2. In the Environment field, enter the name of the environment.

If you select multiple events, all events must be active in that environment.

3. Perform one of the following actions:
   – Select a single event from the Event List field and click Generate Schema(s).

     A file chooser dialog screen appears. To save the generated schema, navigate to the appropriate directory, enter a name for the generated schema in the File Name field, and click Save. If you do not want to save the generated schema, click Cancel.

   – Select two or more events from the Event List field, and click Generate Schema(s).

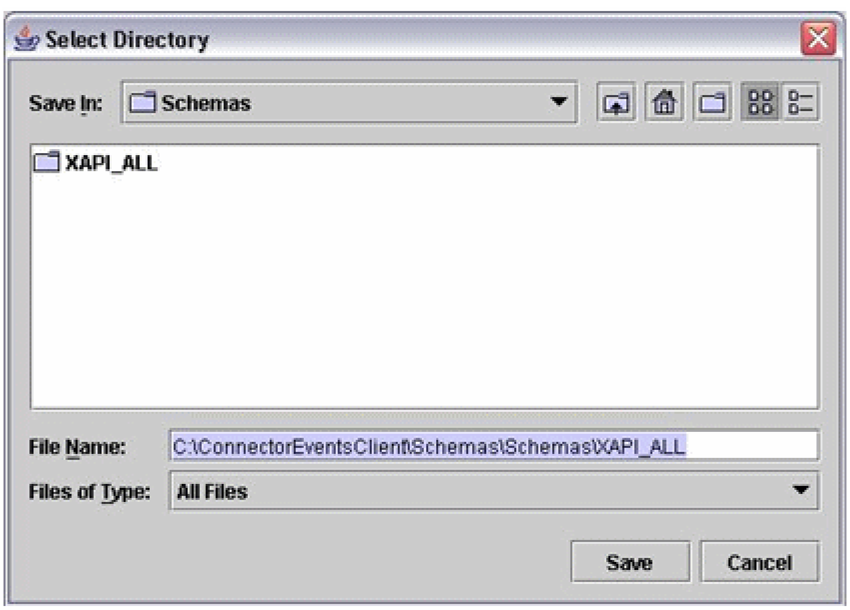
     A Select Directory screen appears. To save the schemas for all of the events, indicate the directory path (use the full path name, for example, C:\ConnectorEventsClient\Schemas), and click Save. The utility saves the schema file for each selected event as E1_EventType.xsd, for example, E1_RTSOOUT.xsd. If you do not want to save the generated schemas, click Cancel.

14.11.3.6 Generating Event Schema for All the Events of a Selected Event Category

You can generate schemas for all of the events within an event category. The events must be active in the environment. On the Event Schema Generator screen, you select the event category and then click Generate All Schemas. A dialog screen named Select Directory appears. You indicate the full path name of the directory where you want to store the schema. The utility saves each schema file as E1_EventType.xsd, for example, E1_RTSOOUT.xsd.

This screen is the Select Directory dialog screen for saving all of the events in a category:

Figure 14–11 Select Directory screen for all events within a category.



14.11.3.7 Generating Header Schema

To generate header schema:

1. Click the Header Schema button and provide the file path with file name to save the header schema.
2. After generating the header schema, the Schema Generation Utility displays the header schema.

14.11.4 Troubleshooting the Schema Generation Utility

This table provides resolutions for problems that might occur:

Problem	Resolution
An error message appears after sign-on.	Ensure that all the given credentials (user name, password, environment, and role) are correct.
C:\SchemaGenUtil\logs directory is getting full of files. Can some of the .log and/or .xml files in that directory be deleted?	Delete any files in that directory at any time. If the Schema Generation Utility application is running, some of the files may be locked.
An error message that you do not understand appears in the Error Messages field.	Look at C:\SchemaGenUtil\logs directory for the jasdebug_date.log file that corresponds to the appropriate date. Often a more explanatory error message can be found in this log file.
Specification not found error is displayed in the Error Messages field.	Verify that the selected event is active and available in the environment that you indicated. This type of error message will read similar to this: Spec not found for requested template: EventCategory: EventType

15

Using Real-Time Events - Guaranteed

This chapter contains the following topics:

Note: This chapter is applicable only if you use guaranteed event delivery. Guaranteed event delivery is available when you use JD Edwards EnterpriseOne Tools 8.94 with JD Edwards EnterpriseOne Applications 8.11 or JD Edwards EnterpriseOne Tools 8.95 and later tools releases with JD Edwards EnterpriseOne Applications 8.10, 8.11, and later EnterpriseOne Applications releases.

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

15.1 Understanding Real-Time Events - Guaranteed

A real-time event is a notification that a business transaction has occurred in JD Edwards EnterpriseOne. You can use a JD Edwards EnterpriseOne HTML client to generate a real-time event on the JD Edwards EnterpriseOne server. Real-time events can be used for both synchronous and asynchronous processing.

An example of synchronous processing is to use real-time events to update an auction site that uses JD Edwards EnterpriseOne as a back-end solution. A user enters a new item for auction, which triggers a transaction into the JD Edwards EnterpriseOne system. The system captures the transaction and sends a notification to an interoperability server that communicates the information to a web engine to update the HTML pages so that all of the auction users can see the new item.

You can also use real-time event generation for asynchronous processing. For example, an online store sends orders to different vendors (business to business), captures the transactions, and enters the orders into the vendors' systems. A user buys a book. The vendor enters a purchase order to the book publisher and sends a notification to the shipping company to pick up the book and deliver it. The book order can be completed as a purchase order transaction with JD Edwards EnterpriseOne, but the shipping request requires that the data is packaged into a commonly agreed-upon format for the shipping company to process.

15.2 Generating Real-Time Events

This section provide an overview about generating real-time events and discusses:

- Real-time event APIs.
- Example code for creating events.

15.2.1 Understanding Real-Time Event Generation

Events can be one of these:

- Single Event

  Contains one partial event. A single event is useful if the receiver requires that events be generated per system call. You can also use single events with different event types.

- Aggregate Event

  Contains multiple partial events. An aggregate event is useful if the receiver requires a document that contains multiple events. For example, a supply chain solution might want the complete sales order provided as one event that contains multiple partial events.

- Composite Event

  Contains only single events. Composite events are useful if the customer has multiple receivers, some requiring single events and some requiring a complete event similar to an aggregate event.

15.2.2 Using Real-Time Event APIs

These APIs are available for you to generate real-time events:

- jdeIEO_EventInit
- jdeIEO_EventAdd
- jdeIEO_EventGetCount
- jdeIEO_EventGetData
- jdeIEO_EventFreeData
- jdeIEO_EventFinalize
- jdeIEO_CreateSingleEvent
- jdeIEO_IsEventTypeEnabled

15.2.3 Interoperability Event Interface Calls Sample Code

These steps and the accompanying example code illustrate how to create a single event:

1. Design the data structure for the real-time event.

```
typedef struct tagDSD55RTTEST
   {
   char    szOrderCo[6];
   char    szBusinessUnit[13];
   char    szOrderType[3];
   MATH_NUMERIC   mnOrderNo;
   MATH_NUMERIC   mnLineNo;
```

```
   JDEDATE     jdRequestDate;
   char    szItemNo[27];
   char    szDescription1[31];
   MATH_NUMERIC   mnQtyOrdered;
   MATH_NUMERIC   mnUnitPrice;
   MATH_NUMERIC   mnUnitCost;
   char    szUserID[11];
   } DSD55RTTEST, *LPDSD55RTTEST;
```

2. Define the data structure object in the business function header file.
3. Modify the business function source to call jdeIEO_CreateSingleEvent.

```
JDEBFRTN(ID) JDEBFWINAPI RealTimeEventsTest (LPBHVRCOM lpBhvrCom,
LPVOID lpVoid, LPDSD55REALTIME lpDS)
{
/* Define Data Structure Object */
DSD55RTTEST  zRTTest   = {0};
   IEO_EVENT_RETURN  eEventReturn   = eEventCallSuccess;
IEO_EVENT_ID  szEventID   ={0};
()Populate required members

/* Now call the API */
   szEventID = jdeIEO_CreateSingleEvent { lpBhvrCom,
      "RealTimeEventsTest",
      "JDERTOUT",
      "SalesOrder",
      "D55RTTEST",
   &zRTTest,
   sizeof(zRTTest),
   0,
   &eEventReturn   };

/* Error in jdeFeedCallObjectEvent is not a critical error
and should only be treated as a warning */
   if( eEventReturn != eEventCallSuccess )
{
   /* LOG the Warning and return */
   return ER_WARNING;
```

This sample code illustrates how to create an aggregate event:

```
DSD55RTTEST zD55TEST01 = {0};
DSD55RTTEST zD55TEST02 = {0};
DSD55RTTEST zD55TEST03 = {0};
IEO_EVENT_RETURN eEventReturn = eEventCallSuccess;
IEO_EVENT_ID szEventID;

szEventID = jdeIEO_EventInit (lpBhvrCom, eEventAggregate, "MyFunction1",
 "JDESOOUT", "EventScope1", 0, &eEventReturn);
eEventReturn = jdeIEO_EventAdd (lpBhvrCom, szEventID, "MyFunction2", NULL,
 "D55TEST01", &zD55TEST01, sizeof(zD55TEST01),0);
eEventReturn = jdeIEO_EventAdd (lpBhvrCom, szEventID, "MyFunction3", NULL,
 "D55TEST02", &zD55TEST02, sizeof(zD55TEST02),0);
eEventReturn = jdeIEO_EventAdd (lpBhvrCom, szEventID, "MyFunction3", NULL,
 "D55TEST03", &zD55TEST03, sizeof(zD55TEST03),0);
eEventReturn = jdeIEO_EventFinalize (lpBhvrCom, szEventID,"MyFunction4",0);
```

This sample code illustrates how to create a composite event:

```
IEO_EVENT_RETURN    eEventReturn    = 0;
    IEO_EVENT_ID szEventID;
```

```
  eEventReturn = eEventCallSuccess;
  szEventID = jdeIEO_EventInit (lpBhvrCom, eEventComposite, "MyFunction1",
"JDESOOUT","EventScope1",0,&eEventReturn,0);
  eEventReturn = jdeIEO_EventAdd ( lpBhvrCom, szEventID, "MyFunction2",
"SODOCBEGIN", "D55TEST01", &zD55TEST01, sizeof(zD55TEST01),0);
  eEventReturn = jdeIEO_EventAdd ( lpBhvrCom, szEventID, "MyFunction3",
"SOITEMADD", "EventScope3", "D55TEST02", &zD55TEST02, sizeof(zD55TEST02),0);
  eEventReturn = jdeIEO_EventFinalize (lpBhvrCom, szEventID, "MyFunction4",0);
```

Errors that are returned by the system calls might not be critical enough to stop the business process. The system flags non-critical errors as warnings and logs them in the log file.

16

Using XAPI Events - Guaranteed

This chapter contains the following topics:

Note: This chapter is applicable only if you use guaranteed events delivery. Guaranteed event delivery is available when you use JD Edwards EnterpriseOne Tools 8.94 with JD Edwards EnterpriseOne Applications 8.11 or JD Edwards EnterpriseOne Tools 8.95 and later tools releases with JD Edwards EnterpriseOne Applications 8.10, 8.11, and later EnterpriseOne Applications releases.

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

16.1 Understanding XAPI Events - Guaranteed

XAPI is a JD Edwards EnterpriseOne service that captures transactions as the transaction occurs and then calls third-party software, end users, and other JD Edwards systems to obtain a return response. A XAPI event is very similar to a real-time event and uses the same infrastructure to send an event. The difference between a real-time event and a XAPI event is that the subscriber to a XAPI event returns a reply to the originator. The XAPI event contains a set of structured data that includes a unique XAPI event name and a business function to be invoked upon return. Like real-time events, XAPI events can be generated on a JD Edwards EnterpriseOne server using a JD Edwards EnterpriseOne HTML client. XAPI events also can be generated by a third-party system and sent to a JD Edwards EnterpriseOne system for a response.

The XAPI structure sends outbound events and receives replies. An event is first generated by the XAPI originator and then sent to a separate system, the XAPI executor, for processing. The XAPI executor then sends a response back to the XAPI originator. The XAPI structure provides for these three possibilities of originator and executor combinations:

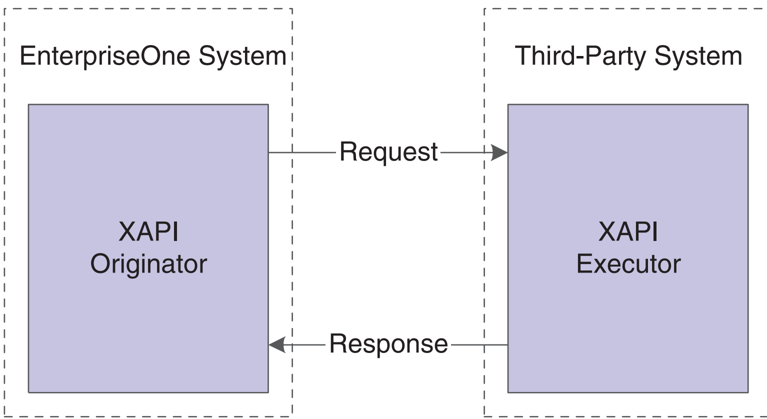
- JD Edwards EnterpriseOne to third-party.
- Third-party to JD Edwards EnterpriseOne.
- JD Edwards EnterpriseOne to JD Edwards EnterpriseOne.

When you use JD Edwards EnterpriseOne-to-EnterpriseOne events processing, you must map business functions and APIs.

16.1.1 JD Edwards EnterpriseOne to Third-Party

This diagram shows a logical representation of the XAPI process from JD Edwards EnterpriseOne to a third-party system:

Figure 16–1 JD Edwards EnterpriseOne to a third-party system XAPI even



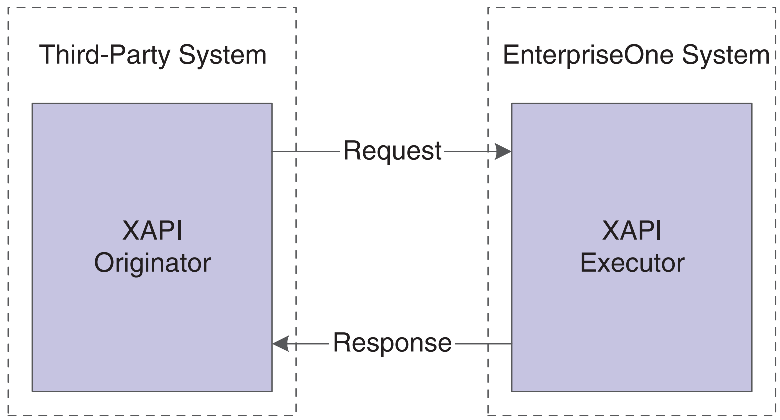
In summary:

1. JD Edwards EnterpriseOne (XAPI originator) sends a request.
2. The request is sent to a third-party system.
3. The third-party system (XAPI executor) processes the request and sends a response back to the XAPI originator.

16.1.2 Third-Party to JD Edwards EnterpriseOne

This diagram shows a logical representation of the XAPI process from a third-party system to JD Edwards EnterpriseOne:

Figure 16–2 Third-party system to JD Edwards EnterpriseOne XAPI event



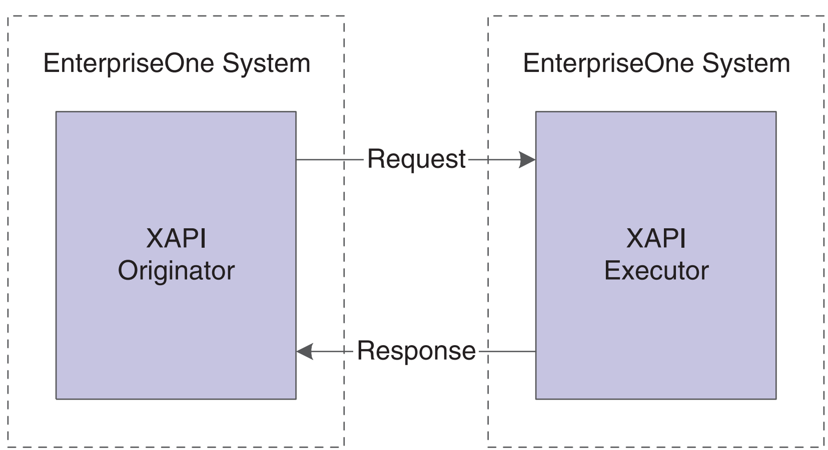
In summary:

1. The third-party system (XAPI originator) sends a request using the JD Edwards EnterpriseOne XAPI request form.
2. The request is sent to JD Edwards EnterpriseOne.
3. JD Edwards EnterpriseOne (XAPI executor) processes the request and sends a response back to the XAPI originator.

16.1.3 JD Edwards EnterpriseOne-to-EnterpriseOne

This diagram shows a logical representation of the XAPI process from one JD Edwards EnterpriseOne system to another JD Edwards EnterpriseOne system:

Figure 16–3 JD Edwards EnterpriseOne-to-EnterpriseOne XAPI event



In summary:

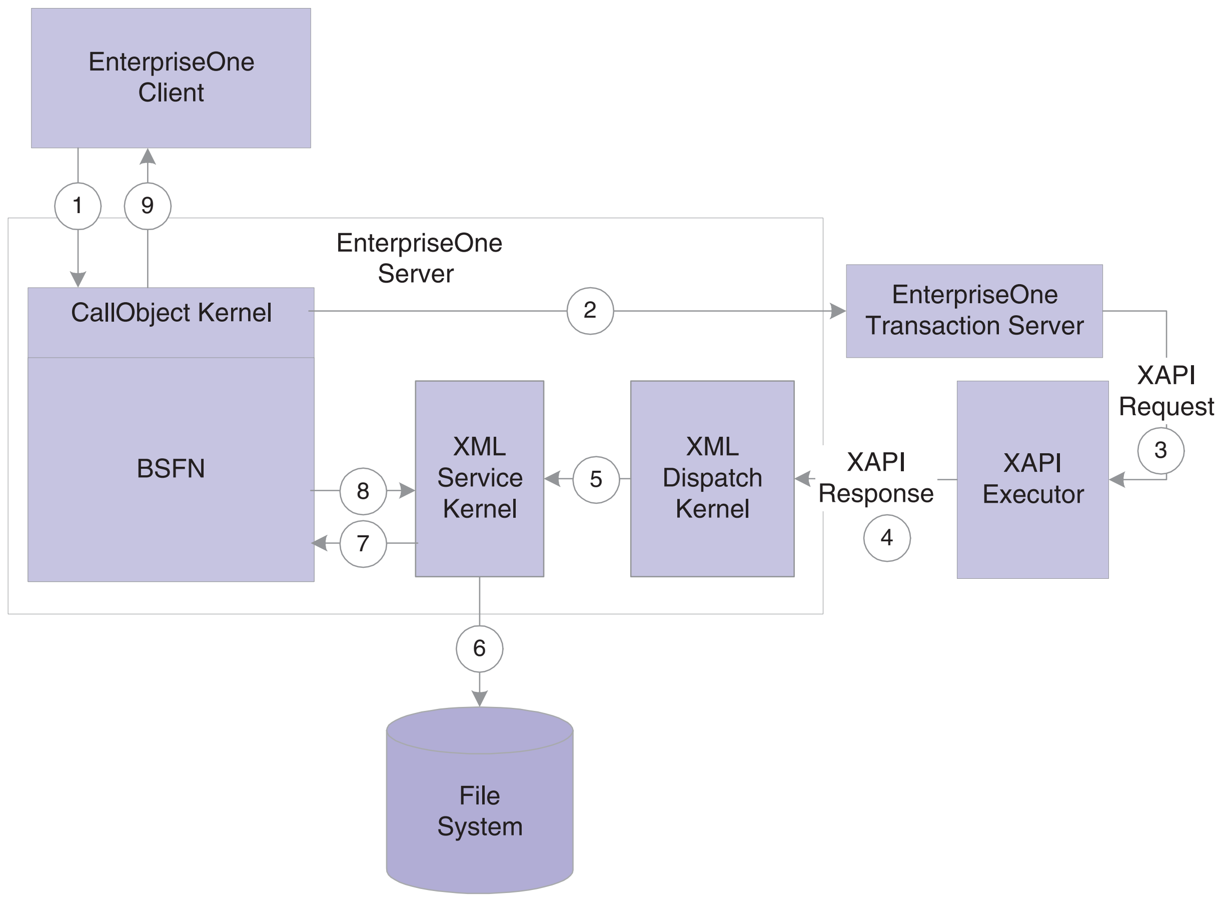
1. The first JD Edwards EnterpriseOne system (XAPI originator) sends a request.
2. The request is sent to a second JD Edwards EnterpriseOne system, which might share the same or different environment as the first JD Edwards EnterpriseOne system.
3. The second JD Edwards EnterpriseOne system (XAPI executor) processes the request and sends a response back to the first JD Edwards EnterpriseOne system (XAPI originator).

4. The first JD Edwards EnterpriseOne system (XAPI originator) processes the response.

16.2 Using JD Edwards EnterpriseOne as a XAPI Originator

This diagram illustrates the flow of a XAPI event when JD Edwards EnterpriseOne functions as the XAPI originator:

Figure 16–4 JD Edwards EnterpriseOne as XAPI originator



In summary:

1. Within the Sending the XAPI Request area in the illustration, a JD Edwards EnterpriseOne client calls a business function on the JD Edwards EnterpriseOne server.
2. The business function uses XAPI APIs to create the XAPI request.

   The CallObject kernel in which the XAPI APIs are executing creates the XAPI request data, adding the callback function. If the XAPI executor is another JD Edwards EnterpriseOne system, the host and port of the JD Edwards EnterpriseOne server that is functioning as the XAPI originator is added to the data. The data is then sent to the Transaction server.
3. The Transaction server sends the document to the subscriber, which is the XAPI executor.

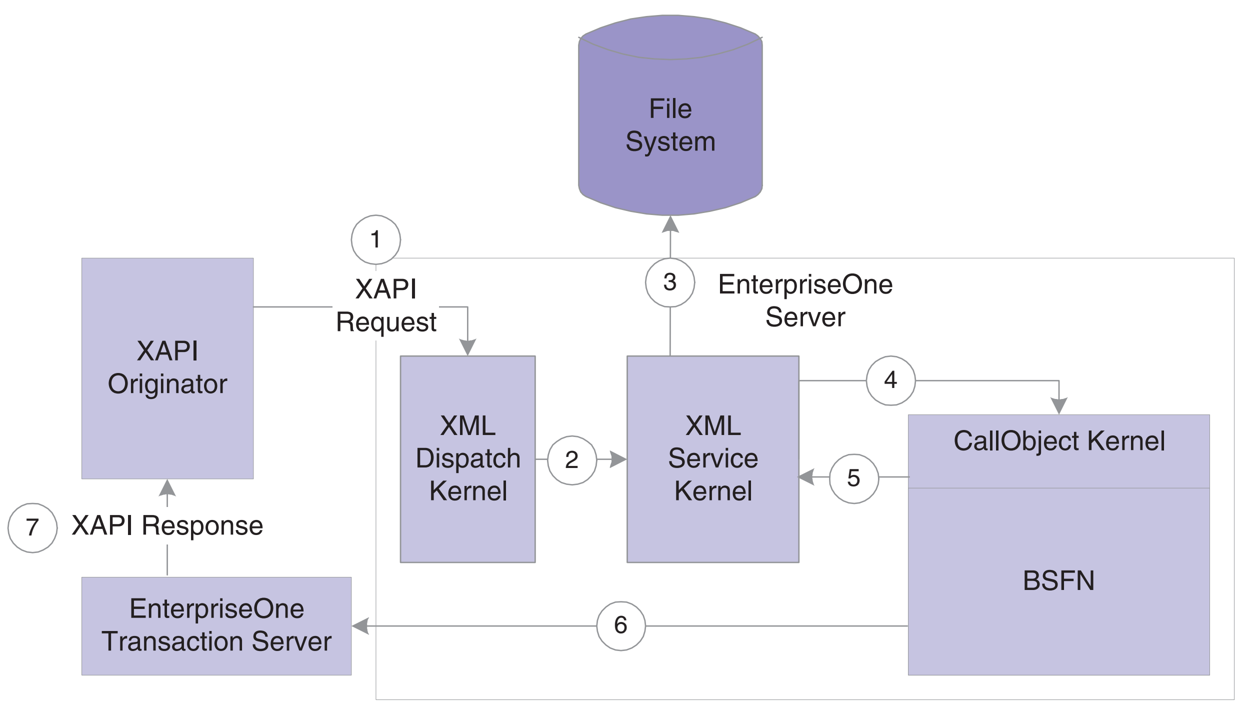
If the XAPI executor is another JD Edwards EnterpriseOne system, the document is sent through JDENET.

4. Within the Receiving the XAPI Response area in the illustration, the XAPI XML response document is sent by the XAPI executor through JDENET to the XML Dispatch kernel of the XAPI executor.
5. The XML Dispatch kernel receives the response XML document and sends the response to the XML Service kernel.
6. The XML Service kernel stores the response document and creates a file handle.
7. The XML Service kernel invokes the callback business function with the file handle.
8. The business function parses the response document using XAPI APIs, which use the XML Service kernel to load the document into memory.
9. The business function uses XAPI APIs to process the response and send it to the JD Edwards EnterpriseOne client.

16.3 Using JD Edwards EnterpriseOne as a XAPI Executor

This diagram illustrates the flow of a XAPI event when JD Edwards EnterpriseOne functions as the XAPI executor.

Figure 16–5 JD Edwards EnterpriseOne as XAPI executor



In summary:

1. Within the Receiving the XAPI Request area of the illustration, the XAPI originator sends the XAPI XML request document to the XML Dispatch kernel through JDENET.
2. The XML Dispatch kernel receives the document and sends the event request and routing information to the XML Service kernel.

3. The XML Service kernel stores the document and creates a file handle for the XAPI request.

   The XML kernel also creates XML-based routing information. The XML Service kernel uses the F907012 table to find the business function that will process the request.

4. The XML Service kernel invokes the business function with the XML request handle and the routing information handle.
5. The business function uses XAPI APIs to parse and process the document. XAPI APIs load the XAPI XML request document into memory.
6. The business function processes the XAPI event request.

   The business function also creates a XAPI response. The message type for the response must be `xapicallmethod`. The business function also passes the routing information handle.

7. Within the Sending the XAPI Response area of the illustration, the business function uses XAPI APIs to send the XAPI response data including the routing information, to the Transaction server.
8. The Transaction server creates the XAPI XML response document and uses the routing information to send the response document to the XAPI originator.

   If the XAPI originator is another JD Edwards EnterpriseOne system, the document is sent through JDENET.

16.4 Working with JD Edwards EnterpriseOne and Third-Party Systems

This section provides an overview of XAPI processing and discusses:

- XAPI outbound request APIs.
- XAPI outbound request API usage code samples.
- XAPI Inbound response APIs.
- XAPI inbound response API usage code samples.

See Also:

- *JD Edwards EnterpriseOne Tools API Reference Guide* on My Oracle Support.

16.4.1 Understanding XAPI Processing between JD Edwards EnterpriseOne and Third-Party Systems

You can use XAPI processing to capture JD Edwards EnterpriseOne transactions as the transaction occurs, and then call third-party software to obtain a return response. In this scenario, JD Edwards EnterpriseOne is the originator, and the third-party system is the executor.

16.4.2 XAPI Outbound Request APIs

These APIs are available for you to generate a XAPI outbound request:

- jdeXAPI_Init
- jdeXAPI_Add
- jdeXAPI_Finalize

- jdeXAPI_Free
- jdeXAPI_SimpleSend
- jdeXAPI_ISCallTypeEnabled
- jdeXAPI_CALLS_ENABLED

16.4.3 XAPI Outbound Request API Usage Code Sample

This code sample illustrates how to create a XAPI outbound request:

```
/* Header files required */

#include <B4205010.h>

/*************************/
      BOOL bXAPIInUse, bExit;
#ifdef jdeXAPI_CALLS_ENABLED
      XAPI_CALL_ID ulXAPICallID = 0;
      XAPI_CALL_RETURN eXAPICallReturn = eEventCallSuccess;
#endif
      DSD4205010A dsD4205010A = {0}; /*Query Header*/
      DSD4205010B dsD4205010B = {0}; /*Query Detail*/
#ifdef jdeXAPI_CALLS_ENABLED
      if(jdeXAPI_IsCallTypeEnabled("XAPIOPOUT") && jdeXAPI_IsCallTypeEnabled
("XAPIOPIN") )
      {
            bXAPIInUse = TRUE;
      }
#endif
      /*-----------------------------------------------------*/
      /* Call XAPIInit */
#ifdef jdeXAPI_CALLS_ENABLED
      if(bXAPIInUse == TRUE)
      {
            ulXAPICallID = jdeXAPI_Init( lpBhvrCom, "SendOrderPromiseRequest",
 "XAPIOPOUT", NULL, &eXAPICallReturn);
            if (eXAPICallReturn != eEventCallSuccess)
            {
                  bExit = TRUE;
            }
      }
#endif
      /*-------------------------------------------------*/
      /* Adding Header Information */
#ifdef jdeXAPI_CALLS_ENABLED
      if(bXAPIInUse == TRUE)
      {
            eXAPICallReturn = jdeXAPI_Add( lpBhvrCom, ulXAPICallID,
"SendOrderPromiseRequest", "D4205010A", &dsD4205010A,
sizeof(DSD4205010A));
            if (eXAPICallReturn != eEventCallSuccess)
            {
                  bExit = TRUE;
            }
      }
#endif
      /*-------------------------------------------------*/
      /* Loading Detail Information */
#ifdef jdeXAPI_CALLS_ENABLED
```

```
      if(bXAPIInUse == TRUE)
      {
            eXAPICallReturn = jdeXAPI_Add( lpBhvrCom, ulXAPICallID,
"SendOrderPromiseRequest", "D4205010B", &dsD4205010B,
sizeof(DSD4205010B));
            if (eXAPICallReturn != eEventCallSuccess)
            {
                  bExit = TRUE;
            }
      }
#endif
#ifdef jdeXAPI_CALLS_ENABLED
      if(bXAPIInUse == TRUE)
      /*--------------------------------------------------*/
      /* Finalize */
      {
            eXAPICallReturn = jdeXAPI_Finalize( lpBhvrCom, ulXAPICallID,
"SendOrderPromiseRequest", "OrderPromiseCallback");
            if (eXAPICallReturn != eEventCallSuccess)
            {
                  bExit = TRUE;
            }
      }
#endif
#ifdef jdeXAPI_CALLS_ENABLED
      if (eXAPICallReturn != eEventCallSuccess)
      {
      /*--------------------------------------------------*/
      /* CleanUp */
            if(bXAPIInUse == TRUE)
            {
                  jdeXAPI_Free( lpBhvrCom, ulXAPICallID,
"SendOrderPromiseRequest");
            }
      }
#endif
```

16.4.4 XAPI Inbound Response APIs

These APIs are available for you to read an inbound XAPI response:

- jdeXML_GetDSCount
- jdeXML_GetDSName
- jdeXML_ParseDS
- jdeXML_DeleteXML

16.4.5 XAPI Inbound Response API Usage Code Sample

This code sample illustrates how the business function uses the XML Service APIs to read and parse the XML data:

```
#include <B4205030.h>

      int iCurrentRecord;
      int iHeaderCount;
      int iRecordCount;
      NID nidDSName;
      DSD4205030A dsD4205030A = {0};
```

```
      DSD4205030B dsD4205030B = {0};
#ifdef jdeXAPI_CALLS_ENABLED
      if(jdeXAPI_IsCallTypeEnabled("XAPIOPOUT") &&      jdeXAPI_IsCallTypeEnabled
("XAPIOPIN") )
      {
            iRecordCount = jdeXML_GetDSCount(lpDS->szXMLHandle);
            if (iRecordCount > 0)
            {
                  for (iCurrentRecord = 0; iCurrentRecord < iRecordCount;
iCurrentRecord++)
                  {
                        jdeXML_
GetDSName(lpDS->szXMLHandle,iCurrentRecord,nidDSName);
                        if (jdestrcmp(nidDSName,(const char*)"D4205030A") ==
0)//mod
                        {
                              jdeXML_ParseDS(
lpDS->szXMLHandle,iCurrentRecord,&dsD4205030A,
sizeof(DSD4205030A));
                        }
                        else
                        {
                              jdeXML_ParseDS(
lpDS->szXMLHandle,iCurrentRecord,&dsD4205030B,
sizeof(DSD4205030B));

                        }
                  }
            }
            if (iCurrentRecord == iRecordCount)
            {
                  jdeXML_DeleteXML(lpDS->szXMLHandle);
            }
      }
#endif
```

16.5 Using JD Edwards EnterpriseOne-to-EnterpriseOne Connectivity

This section provides an overview of the JD Edwards EnterpriseOne-to-EnterpriseOne connectivity for XAPI events and discusses:

- XAPI outbound request handling APIs.
- XAPI outbound request parsing API usage sample code.
- XAPI inbound response generation APIs.
- XAPI inbound response parsing API usage sample code.
- XAPI error handling APIs.

See Also:

- *JD Edwards EnterpriseOne Tools API Reference Guide* on My Oracle Support.

16.5.1 Understanding JD Edwards EnterpriseOne-to-EnterpriseOne Connectivity

The XAPI structure provides the capability for two different JD Edwards EnterpriseOne systems to communicate with each other. The first JD Edwards EnterpriseOne system (XAPI originator) generates a XAPI request (event). Instead of the request being distributed to a third-party system, JDENET sends the request to a second JD Edwards EnterpriseOne system. A JD Edwards EnterpriseOne to JD Edwards EnterpriseOne XAPI event must be sent through a subscriber with the JDENET transport type. The second JD Edwards EnterpriseOne system (XAPI executor) processes the event and returns a response to the first JD Edwards EnterpriseOne system (XAPI originator).

16.5.1.1 Modify Element Name for XML Documents

Before XAPI event processing, any document that was sent from JD Edwards EnterpriseOne was considered to be a response document, and any document coming in to JD Edwards EnterpriseOne was considered to be a request document. However, with XAPI, request documents are generated by the JD Edwards EnterpriseOne originating system and can be sent to a JD Edwards EnterpriseOne executor system. Response documents are generated and sent out by the JD Edwards EnterpriseOne executor system and received by the JD Edwards EnterpriseOne originating system. To support XAPI and to enable the XML dispatch kernel to be able to distinguish between a response and reply, JD Edwards created these type attributes to be used with the jdeResponse element:

Element and Type Attribute	Description
jdeResponse=RealTimeEvent	Use this element and attribute to identify a XAPI request that is sent from the JD Edwards EnterpriseOne originating system and sent to the JD Edwards EnterpriseOne executor system.
jdeResponse=xapicallmethod	Use this element and attribute to identify a XAPI response that is sent from the JD Edwards EnterpriseOne executor system and sent to the JD Edwards EnterpriseOne originating system.

When the XML Dispatch kernel receives a document with the jdeResponse element and a RealTimeEvent or xapicallmethod type attribute, XML Dispatch sends the document to the XML Service kernel. XML Service can distinguish a response or a reply based on the type attribute that is associated with the jdeResponse element and then processes the document appropriately.

16.5.1.2 Security for Originator and Executor

Access to the JD Edwards EnterpriseOne originator and JD Edwards EnterpriseOne executor systems is based on:

- Security token
- Environment
- Role

The JD Edwards EnterpriseOne originating system verifies that the security information is valid and creates an hUser object with an encrypted token to send to the JD Edwards EnterpriseOne executor. Encryption APIs (jdeEncypher and jdeDecypher) are used to encrypt and decode the password. The security information is sent in the XAPI request XML document.

> **Note:** The user ID, password, environment, and role must be the same on both JD Edwards EnterpriseOne systems (originator and executor).

16.5.1.3 Error Processing for Originator and Executor

You might encounter these two errors during XAPI error processing between two JD Edwards EnterpriseOne systems:

Type of Error	Explanation
Business-related errors	The business function or the business function specs cannot be found.
System errors	These errors occur in other parts of the system (for example, message delivery failure).

The system handles XAPI error processing for business-related errors in these ways:

- XAPI logs business-related errors in the JD Edwards EnterpriseOne server log, and the errors are delivered as part of the XAPI reply
- XAPI APIs parse business errors from the response document.
- XAPI logs all information that is available about the error in the JD Edwards EnterpriseOne server log.

16.5.2 XAPI Outbound Request Handling APIs

These outbound request handling APIs are available for you to generate a JD Edwards EnterpriseOne-to-EnterpriseOne XAPI outbound request:

- jdeXMLRequest_GetDSCount
- jdeXMLRequest_GetDSName
- jdeXMLRequest_ParseDS
- jdeXMLRequest_DeleteXML
- jdeXMLRequest_ParseNextDSByName
- jdeXMLRequest_PrepareDSListForIterationByName

16.5.3 XAPI Outbound Request Parsing API Usage Sample Code

This code sample shows the API usage for parsing an outbound request by the JD Edwards EnterpriseOne XAPI executor:

```
#include <jde.h>

#define b0000310_c


/*****************************************************************************
*    Source File:  b0000310
*
*    Description:  Company Real Time Notification Outbound Wrapper Source File
*
*****************************************************************************/

#include <b0000310.h>
#include <B4206030.h>
#include <B4206000.h>
/**************************************************************************
*  Business Function:  CompanyRealTimeWrapper
*
*        Description:  Company Real Time Notification Outbound Wrapper
```

```
*
*        Parameters:
*          LPBHVRCOM          lpBhvrCom    Business Function Communications
*          LPVOID             lpVoid       Void Parameter - DO NOT USE!
*          LPDSD0000310A       lpDS         Parameter Data Structure Pointer
*
*************************************************************************/

      int iXMLRecordCount = 0;
      int iCurrentRecord = 0;
      NID nidDSName;
      ID idReturnValue = ER_SUCCESS;
      ID idSORecordCount = ER_ERROR; /*Return Code*/
      LPDSD4206000A lpDS;
      int lpmnJobNumber;

      MATH_NUMERIC mnBatchNumber = {0};
      unsigned long lBatchNumber = {0};
      DSD4206030A dsD4206030A = {0};

      /* CacheProcessInboundDemandRequest B4206030.c */
      DSD4206000I dsD4206000I = {0};

      /* Demand scheduling inbound DSTR */
      iXMLRecordCount = jdeXMLRequest_GetDSCount(lpDS->szXMLHandle);
      if( iXMLRecordCount > 0)
      {
            for ( iCurrentRecord = 0; iCurrentRecord < iXMLRecordCount;
iCurrentRecord++)
            {
                  memset((void *)(&dsD4206000I), (int)(_J('\0')),
sizeof(DSD4206000I));
                  memset((void *)(nidDSName), (int)(_J('\0')), sizeof(NID));
                  if(jdeXMLRequest_
GetDSName(lpDS->szXMLHandle,iCurrentRecord,nidDSName))
                  {
                        /* Retrieving data*/
                        if (jdeStricmp(nidDSName, (const JCHAR *)_J("D40R0180B"))
== 0)
                        {
                              if (jdeXMLRequest_
ParseDS(lpDS->szXMLHandle,iCurrentRecord,
&dsD4206000I,sizeof(DSD4206000I)))
                              {
                                    /* Get next number for the batch number of the
inbound INVRPT
record*/
                                    if ( dsD4206000I.cInventoryAdvisement == _
J('1'))
                                    {
                                          lBatchNumber = JDB_
GetInternalNextNumber();
                                          LongToMathNumeric(lBatchNumber,
&mnBatchNumber);

FormatMathNumeric(dsD4206000I.szBatch,&mnBatchNumber);
                                    }
                                    /* Setup cancel flag for pending delete record
*/
                                    if ( dsD4206000I.cPendingDelete == _J('1'))
```

```
                                    {
                                          /* Flag set as 1 for any cancel demand
record */
                                          dsD4206000I.cCancelFlag = _J('1');
                                    }
                                    else
                                    { /* Flag set as 9 for any non cancel demand
record */
                                          dsD4206000I.cCancelFlag = _J('9');
                                    }
                                    /* Load parms for cache */
                                    //memset((void *)(&dsD4206030A), (int)(_
J('\0')),
sizeof(DSD4206030A));
                                    I4206000_LoadParmsToCache(&dsD4206000I,
&dsD4206030A);
                                    MathCopy(&dsD4206030A.mnJobnumberA,
lpmnJobNumber);
                                    /* Add the DSTR to cache */
                                    idReturnValue = jdeCallObject( _
J("CacheProcessInboundDemand
Request") ,(LPFNBHVR)NULL ,lpBhvrCom ,lpVoid ,(LPVOID)&dsD4206030A,
(CALLMAP *)NULL, (int)0,(JCHAR*)NULL ,(JCHAR*)NULL ,(int)0 );
                                    /* Write XML DSTR to cache fail */
                                    if (idReturnValue == ER_ERROR)
                                    {
                                          jdeErrorSet(lpBhvrCom, lpVoid, (ID)0, _
J("032E"), (LPVOID)NULL);
                                    }
                              }
                              else
                              { /* warning XML parse fail */
                                    jdeErrorSet(lpBhvrCom, lpVoid, (ID)0, _
J("40R46"), (LPVOID) NULL);
                              }
                        } /* end if */
                  }/* end if DS name */
            }/* end for - looping all matching XML DSTR */
            /* Ensure there is at least one record */
            idSORecordCount = ER_SUCCESS;
      }/*if( iXMLRecordCount > 0) */
      return idSORecordCount;
```

16.5.4 XAPI Inbound Response Generation APIs

These outbound request handling APIs are available for you to generate a JD Edwards EnterpriseOne-to-EnterpriseOne XAPI outbound request:

- jdeXAPIResponse_SimpleSend
- jdeXAPIResponse_Init
- jdeXAPIResponse_Add
- jdeXAPIResponse_Finalize
- jdeXAPIResponse_Free

16.5.5 XAPI Inbound Response Parsing API Usage Sample Code

This code sample shows the API usage for generating an inbound response from the JD Edwards EnterpriseOne XAPI executor to the JD Edwards EnterpriseOne originator:

```
JDEBFRTN (ID) JDEBFWINAPI SendOrderPromiseRequest (LPBHVRCOM lpBhvrCom,
LPVOID lpVoid, LPDSD4205010 lpDS)
{
/*************************************************************
 * Variable declarations
*************************************************************/
  char    cPromisableLine          = ' ';
  int     nHeaderBackOrderAllowed  = ' ';
  HUSER   hUser;
  ID      JDEDBResult              = JDEDB_PASSED;
  BOOL    bExit                    = FALSE;
  BOOL    bB4001040Called          = FALSE;
  BOOL    bXAPIInUse               = FALSE;
  BOOL    bAtLeastOneDetail        = FALSE;

  #ifdef jdeXAPI_CALLS_ENABLED
  XAPI_CALL_ID ulXAPICallID        = 0;
  XAPI_CALL_RETURN eXAPICallReturn = eEventCallSuccess;
  #endif
/*************************************************************
 * Declare structures
*************************************************************/
  DSD4001040   dsD4001040     = {0};
  DSD4205020   dsD4205020     = {0};
  DSD4205040   dsD4205040     = {0}; /* Header Info */
  DSD4205050   dsD4205050     = {0}; /* Detail Info */
  DSD4205010A   dsD4205010A   = {0}; /* Query Header */
  DSD4205010B   dsD4205010B   = {0} /* Query Detail */
  DSD0100042   dsD0100042     = {0};
  LPDSD4205040H  lpDSD4205040H    = (LPDSD4205040H) NULL;
  LPDSD4205050D  lpDSD4205050D    = (LPDSD4205050D) NULL;

/*************************************************************
 * Declare pointers
*************************************************************/
/*************************************************************
 * Check for NULL pointers
*************************************************************/
if ((lpBhvrCom == (LPBHVRCOM) NULL) ||
 (lpVoid == (LPVOID)  NULL) ||
 (lpDS == (LPDSD4205010) NULL))
{
jdeErrorSet (lpBhvrCom, lpVoid, (ID) 0, "4363", (LPVOID) NULL);
return ER_ERROR;
}

/* Retrieving hUser */
JDEDBResult = JDB_InitBhvr (lpBhvrCom, &hUser, (char *)NULL,
JDEDB_COMMIT_AUTO ) ;

if ( JDEDBResult == JDEDB_FAILED )
{
jdeSetGBRError ( lpBhvrCom, lpVoid, (ID) 0, "4363" ) ;
return ER_ERROR ;
}
```

```
/****************************************************************
 * Set pointers
****************************************************************/
/****************************************************************
 * Main Processing
****************************************************************/
 /*----------------------------------------------------*/
 /* Setting Up ErrorCode
      */
 lpDS->cErrorCode = '0';
 /*----------------------------------------------------*/
 /* Determining if XAPI is ready to be used            */

 bXAPIInUse = FALSE;

 #ifdef jdeXAPI_CALLS_ENABLED
 if(jdeXAPI_IsCallTypeEnabled("XAPIOPOUT") &&
     jdeXAPI_IsCallTypeEnabled("XAPIOPIN") )
 {
     bXAPIInUse = TRUE;
 }
 #endif

 /*-----------------------------------------------------*/
 /* Data validation and default values.                 */
 /* When Display Before Accept Mode is on, validate Key  */
 /* Information. Otherwise retrieve it from Header Record*/

 if((lpDS->cDisplayBeforeAcceptMode == '1')   &&
      ((MathZeroTest(&lpDS->mnOrderNumber) == 0) ||
       (IsStringBlank(lpDS->szOrderType))    ||
       (IsStringBlank(lpDS->szOrderCompany))))
 {
      bExit = TRUE;
 }
 else
 {
     MathCopy(&dsD4205040.mnOrderNumber,&lpDS->mnOrderNumber);
     strncpy(dsD4205040.szOrderType,
       lpDS->szOrderType,
        sizeof(dsD4205040.szOrderType));
     strncpy(dsD4205040.szComputerID,
       lpDS->szOrderCompany,
        sizeof(dsD4205040.szOrderCompany));
     dsD4205040.cUseCacheOrWF = lpDS->cUseCacheOrWF;
     strncpy(dsD4205040.szComputerID,
       lpDS->szComputerID,
        sizeof(dsD4205040.szComputerID));
     MathCopy(&dsD4205040.mnJobNumber,&lpDS->mnJobNumber);
     jdeCallObject( "GetSalesOrderHeaderRecord",
          NULL,
          lpBhvrCom, lpVoid,
          (LPVOID)&dsD4205040,
          (CALLMAP *) NULL,
          (int) 0,
          (char *) NULL,
          (char *) NULL,
          (int) 0 ) ;

     lpDSD4205040H = (LPDSD4205040H)jdeRemoveDataPtr(hUser,
```

```
(ulong)dsD4205040.idHeaderRecord);

     if (lpDSD4205040H == NULL)
     {
          bExit = TRUE;
     }
 }

 /*-------------------------------------------------------*/
 /* Set error if exiting at this point                  */
 if (bExit == TRUE)
 {
     lpDS->cErrorCode = '1';
     /* Sales Order Header Not Found */
     strncpy(lpDS->szErrorMessageID,
        "072T",
          sizeof(lpDS->szErrorMessageID));
     if (lpDS->cSuppressError != '1')
     {
        jdeErrorSet (lpBhvrCom, lpVoid, (ID) 0, "072T", (LPVOID) NULL);
     }
 }

 /*-------------------------------------------------------*/
 /* Default Promising Flag is always 1                  */
 lpDS->cDefaultPromisingFlags = 1;
 if (bExit == FALSE)
 {
  /*-------------------------------------------------------*/
  /* Call XAPIInit   */
  #ifdef jdeXAPI_CALLS_ENABLED
  if(bXAPIInUse == TRUE)
  {
     ulXAPICallID = jdeXAPI_Init( lpBhvrCom,
                   SendOrderPromiseRequest,
                "XAPIOPOUT",
                NULL,
                &eXAPICallReturn);
     if (eXAPICallReturn != eEventCallSuccess)
     {
          bExit = TRUE;
     }
  }
  #endif
  if (bExit == FALSE)
  {

     /*-------------------------------------------------*/
     /* Loading Header Information                      */
     I4205010_PopulateQueryHeader(lpDS,&dsD4205010A
       lpDSD4205040H,&dsD0100042,hUser,lpVoid,lpBhvrCom);
     nHeaderBackOrderAllowed = dsD4205010A.nAllowBackorders;

     /*-------------------------------------------------*/
     /* Adding Header Information                       */
     #ifdef jdeXAPI_CALLS_ENABLED
     if(bXAPIInUse == TRUE)
     {
        eXAPICallReturn = jdeXAPI_Add( lpBhvrCom,
                   ulXAPICallID,
```

```
                    "SendOrderPromiseRequest",
                    "D4205010A",
                    &dsD4205010A,
                    sizeof(DSD4205010A));
         if (eXAPICallReturn != eEventCallSuccess)
         {
            bExit = TRUE;
         }
      }
      #endif
   }
  }
  if (bExit == FALSE)
  {

   /*-----------------------------------------------------*/
   /* Loading Detail Information                          */
   MathCopy(&dsD4205050.mnOrderNumber,&lpDS->mnOrderNumber);
   strncpy(dsD4205050.szOrderType,lpDS->szOrderType,
               sizeof(dsD4205050.szOrderType));
   strncpy(dsD4205050.szOrderCompany,lpDS->szOrderCompany,
               sizeof(dsD4205050.szOrderCompany));
   dsD4205050.cUseCacheOrWF = lpDS->cUseCacheOrWF;
   strncpy(dsD4205050.szComputerID,lpDS->szComputerID,
               sizeof(dsD4205050.szComputerID));
   MathCopy(&dsD4205050.mnJobNumber,&lpDS->mnJobNumber);
   if (lpDSD4205040H->cActionCode != 'A')
   {
      dsD4205050.cCheckTableAfterCache = '1';
   }
   else
   {
      dsD4205050.cCheckTableAfterCache = '0';
   }
   jdeCallObject( "GetSalesOrderDetailRecordOP",
      NULL,
      lpBhvrCom, lpVoid,
      (LPVOID)&dsD4205050,
      (CALLMAP *) NULL,
      (int) 0, (char *) NULL,
      (char *) NULL, (int) 0 ) ;

   if (dsD4205050.cRecordFound != '1')
   {
      bExit = TRUE;
      lpDS->cErrorCode = '1';

      /* Sales Order Detail Not Found */
      strncpy(lpDS->szErrorMessageID,"4162",
               sizeof(lpDS->szErrorMessageID));
      if (lpDS->cSuppressError != '1')
      {
         jdeErrorSet (lpBhvrCom, lpVoid, (ID) 0, "4162", (LPVOID) NULL);
      }
   }
   while ((dsD4205050.cRecordFound == '1') && (bExit == FALSE))
   {
      lpDSD4205050D = (LPDSD4205050D)jdeRemoveDataPtr( hUser,
(ulong)dsD4205050.idDetailRecord);
      /* Reset flags */
```

```
cPromisableLine = '0';
bB4001040Called = FALSE;

/*-----------------------------------------------*/
/* Evaluate the Record from F4211 (cDataSource = 2)*/
/*  to find out if we should promise the line      */
/*  else find out from Order Promising Detail.     */

if(dsD4205050.cDataSource == '1')
{
   if (lpDSD4205050D->cOPPromiseLineYN == 'Y')
   {
      cPromisableLine = '1';
   }
}
else if(dsD4205050.cDataSource == '2')
{
   MathCopy ( &dsD4001040.mnShortItemNumber,
        &lpDSD4205050D->mnShortItemNumber);
   strncpy ( dsD4001040.szBranchPlant,
      lpDSD4205050D->szBusinessUnit,
         sizeof(dsD4001040.szBranchPlant));

   jdeCallObject  ( "GetItemMasterDescUOM",
            NULL,
            lpBhvrCom, lpVoid,
             (LPVOID)&dsD4001040,
             (CALLMAP *) NULL,
             (int) 0, (char *) NULL,
             (char *) NULL, (int) 0 ) ;

   bB4001040Called = TRUE;

   cPromisableLine = I4205010_IsLinePromisable(lpBhvrCom,lpVoid,
              hUser,lpDS,lpDSD4205050D, dsD4001040.cStockingType);
}
if (cPromisableLine == '1')
{

  /* Set this flag if at least one promisable */
  /*   detail record exists.                  */
  bAtLeastOneDetail = TRUE;

  if (bB4001040Called == FALSE)
  {
    MathCopy (&dsD4001040.mnShortItemNumber,
         &lpDSD4205050D->mnShortItemNumber);
    strncpy ( dsD4001040.szBranchPlant,
         lpDSD4205050D->szBusinessUnit,
         sizeof(dsD4001040.szBranchPlant));

  jdeCallObject  ( "GetItemMasterDescUOM",
           NULL,
           lpBhvrCom, lpVoid,
           (LPVOID)&dsD4001040,
           (CALLMAP *) NULL,
           (int) 0, (char *) NULL,
           (char *) NULL, (int) 0 ) ;
  }
```

```
  I4205010_PopulateQueryDetail( lpDS,&dsD4205010B,
              lpDSD4205050D,
           &dsD4001040,
           &dsD4205010A,
              &dsD0100042,
           cPromisableLine,
              hUser,
              lpVoid,
           lpBhvrCom);

  #ifdef jdeXAPI_CALLS_ENABLED
  if(bXAPIInUse == TRUE)
  {
    eXAPICallReturn = jdeXAPI_Add( lpBhvrCom,
              ulXAPICallID,
              "SendOrderPromiseRequest",
              "D4205010B",
              &dsD4205010B,
              sizeof(DSD4205010B));
    if (eXAPICallReturn != eEventCallSuccess)
    {
         bExit = TRUE;
    }
  }
  #endif
}

/*--------------------------------------------------*/
/* Fetching the next Detail Record          */
MathCopy(&dsD4205050.mnOrderNumber,&lpDS->mnOrderNumber);
strncpy(dsD4205050.szOrderType,lpDS->szOrderType,
          sizeof(dsD4205050.szOrderType));
strncpy(dsD4205050.szOrderCompany,lpDS->szOrderCompany,
         sizeof(dsD4205050.szOrderCompany));
dsD4205050.cUseCacheOrWF = lpDS->cUseCacheOrWF;
strncpy(dsD4205050.szComputerID,lpDS->szComputerID,
         sizeof(dsD4205050.szComputerID));
MathCopy(&dsD4205050.mnJobNumber,&lpDS->mnJobNumber);
if (lpDSD4205040H->cActionCode != 'A')
{
   dsD4205050.cCheckTableAfterCache = '1';
   }
   else
   {
       dsD4205050.cCheckTableAfterCache = '0';
   }
   jdeCallObject( "GetSalesOrderDetailRecordOP",
           NULL,
           lpBhvrCom, lpVoid,
           (LPVOID)&dsD4205050,
           (CALLMAP *) NULL,
           (int) 0, (char *) NULL,
           (char *) NULL, (int) 0 ) ;
}
if (!bAtLeastOneDetail)
{
   bExit = TRUE;
   lpDS->cErrorCode = '1';
   /* Sales Order Detail Not Found */
   strncpy(lpDS->szErrorMessageID,"4162",
```

```
                 sizeof(lpDS->szErrorMessageID));
         if (lpDS->cSuppressError != '1')
         {
            jdeErrorSet (lpBhvrCom, lpVoid, (ID) 0, "4162", (LPVOID) NULL);
         }
      }
      if (bExit == FALSE)
      {
         #ifdef jdeXAPI_CALLS_ENABLED
         if(bXAPIInUse == TRUE)
         {
            eXAPICallReturn = jdeXAPI_Finalize( lpBhvrCom,
                    ulXAPICallID,
                    "SendOrderPromiseRequest",
                    "OrderPromiseCallback)";
            if (eXAPICallReturn != eEventCallSuccess)
            {
               bExit = TRUE;
            }
         }
         #endif
      }

      /*-----------------------------------------------------*/
      /* Call B4205020 in Add Mode                           */
      if((bExit == FALSE) &&
          (lpDS->cDisplayBeforeAcceptMode != '1') &&
          (lpDS->cUseCacheOrWF == '2'))
      {
          MathCopy(&dsD4205020.mnOrderNumber,&lpDS->mnOrderNumber);
          strncpy(dsD4205020.szOrderType,lpDS->szOrderType,
               sizeof(dsD4205020.szOrderType));
          strncpy(dsD4205020.szOrderCompany,lpDS->szOrderCompany,
               sizeof(dsD4205020.szOrderCompany));
         strncpy(dsD4205020.szComputerID,lpDS->szComputerID,
               sizeof(dsD4205020.szComputerID));
         MathCopy(&dsD4205020.mnJobNumber,&lpDS->mnJobNumber);

         jdeCallObject( MaintainOPWorkFile,
             NULL,
             lpBhvrCom, lpVoid,
             (LPVOID)&dsD4205020,
             (CALLMAP *) NULL,
             (int) 0, (char *) NULL,
             (char *) NULL, (int) 0 ) ;
      }
   }

/*****************************************************************
 * Function Clean Up
*****************************************************************/
  #ifdef jdeXAPI_CALLS_ENABLED
  if (eXAPICallReturn != eEventCallSuccess)
  {
   /*-----------------------------------------------------*/
   /* CleanUp */
   if(bXAPIInUse == TRUE)
   {
    jdeXAPI_Free( lpBhvrCom,
        ulXAPICallID,
```

```
          "SendOrderPromiseRequest");
      }

      lpDS->cErrorCode = '1';
      /* System Error - no reasonable error messages exist. */
      strncpy(lpDS->szErrorMessageID,"018Y",
                   sizeof(lpDS->szErrorMessageID));
      if (lpDS->cSuppressError != '1')
      {
          jdeErrorSet (lpBhvrCom, lpVoid, (ID) 0, "018Y", (LPVOID) NULL);
      }
     }
     #endif
     if(lpDSD4205040H != (LPDSD4205040H)NULL)
     {
        jdeFree((void *)lpDSD4205040H);
     }
     if(lpDSD4205050D != (LPDSD4205050D)NULL)
     {
        jdeFree((void *)lpDSD4205050D);
     }
    return (ER_SUCCESS);
   }
```

16.5.6 XAPI Error Handling APIs

These APIs are used for error handling in the XAPI executor system.

- jdeXML_CheckSystemError

  The check system error API is for system errors. It tells the JD Edwards EnterpriseOne originator system that a system error occurred in the JD Edwards EnterpriseOne executor system:

- jdeXML_GetErrorCount
- jdeXML_SetErrors

  The get error count and set errors APIs are for business errors. These two APIs, when used together, find the number of business errors and then send the errors to the BHVRCOM structure for you to resolve.

16.6 Mapping a Business Function

This section provides an overview of mapping business functions and discusses how to add mapping information.

16.6.1 Understanding how to Map a Business Function

When the JD Edwards EnterpriseOne executor system receives an event from the JD Edwards EnterpriseOne originator, the JD Edwards EnterpriseOne executor needs to know what business function or system API to invoke to process the request. You must map the business function or system API to the XAPI event name. You map business functions and system APIs in the F907012 table. You use the Event Request Definition program (P907012) to map business functions and APIs.

If you are mapping business functions, you enter the name of the business function. If you are mapping APIs, you must enter the name of the API and the library where it is defined. In addition, the signature of the API must be made common, similar to the business function.

Mapping business functions enables you to point a XAPI event to a business function or system API that you wrote. You do not need to modify source code of a business function that JD Edwards delivered to you.

16.6.2 Forms Used to Add Mapping Information

Form Name	FormID	Navigation	Usage
Work With Definition	W907012A	Enter **P907012** in the Fast Path Command Line.	Locate and review existing mappings.
Request Definition	W907012B	On Work With Definition, click Add.	Add or change business function or API mapping for the XAPI event.

16.6.3 Adding Mapping Information

Access the Request Definition form.

Event Name
The name of the event (for example JDERTSOOUT). Some events are part of other events.

BSFN Definition
An option that specifies the type of processing for an event.

API Definition
An option that specifies the type of processing for an event.

When you select the API definition option, the DLL Name field appears on the form.

Function Name
The actual name of the function. It must follow standard ANSI C naming conventions (for example, no space between words).

DLL Name
Specifies the name of the database driver file. This file is specified in the [DB SYSTEM SETTINGS] section of the enterprise server jde.ini file. The file you specify depends upon the platform and the database. Values for specific machines and databases are:

DBDR: IBM i to DB2 for IBM i

JDBNET: IBM i to any other server DBMS

libjdbnet.sl: HP9000 to DB2 for IBM i

libjdbnet.sl: HP9000 to Microsoft SQL Server

libora80.sl: HP9000 to Oracle (Version 8.0) UNIX

libjdbnet.so: RS6000 to DB2 for IBM i

libjdbnet.so: RS6000 to Microsoft SQL Server

libora73.so: RS6000 to Oracle (Version 7.3) UNIX

libora80.so: RS6000 to Oracle (Version 8.0) UNIX

jdbodbc.dll: Intel to DB2 for IBM i =

jdboci32.dll: Intel to Oracle (Version 7.2) NT

jdboci73.dll: Intel to Oracle (Version 7.3) NT

dboci80.dll: Intel to Oracle (Version 8.0) NT

dbodbc.dll: Intel to SQL Server NT

jdbnet.dll: Digital Alpha to DB2 for IBM i

jdboci32.dll: Digital Alpha to Oracle (Version 7.2)

dboci73.dll: Digital Alpha to Oracle (Version 7.3)

jdboci80.dll: Digital Alpha to Oracle (Version 8.0)

jdbodbc.dll: Digital Alpha to SQL Server NT

17

Using Z Events - Guaranteed

This chapter contains the following topics:

- Section 17.1, "Understanding Z Events - Guaranteed"
- Section 17.2, "Z Event Process Flow"
- Section 17.3, "Vendor-Specific Outbound Functions"
- Section 17.4, "Working With Z Events"
- Section 17.5, "Setting Up Data Export Controls"

This chapter provides overviews of Z events, the Z event process, and vendor-specific outbound functions, and discusses how to work with Z events.

Note: This chapter is applicable only if you use guaranteed events delivery. Guaranteed event delivery is available when you use JD Edwards EnterpriseOne Tools 8.94 with JD Edwards EnterpriseOne Applications 8.11 or JD Edwards EnterpriseOne Tools 8.95 and later tools releases with JD Edwards EnterpriseOne Applications 8.10, 8.11, and later EnterpriseOne Applications releases.

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

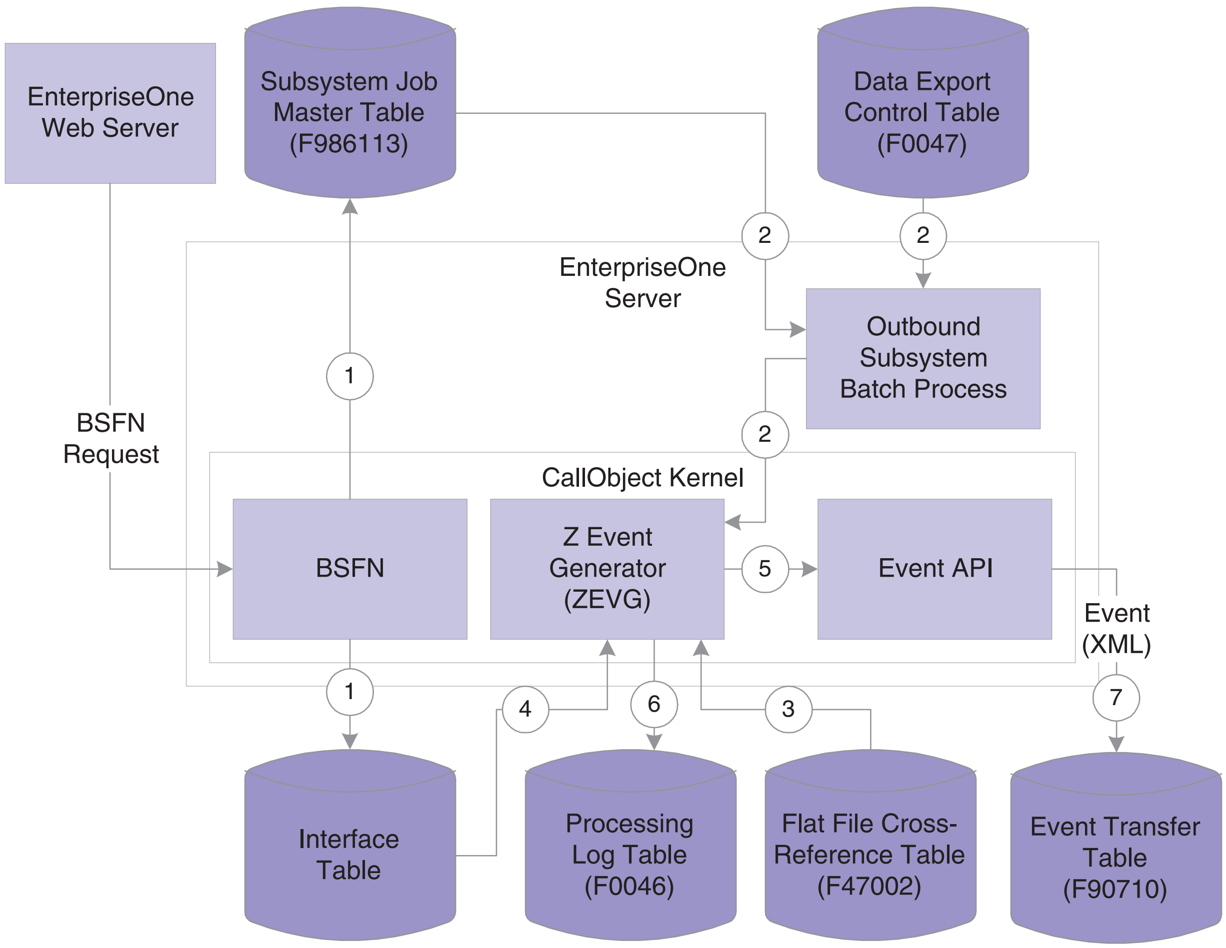
17.1 Understanding Z Events - Guaranteed

A Z event is near real-time notification that an interoperability transaction has occurred. To generate Z events, JD Edwards EnterpriseOne uses the Z event generator and the existing interface table infrastructure. You can use the existing JD Edwards EnterpriseOne interface tables, or you can build customized interface tables as long as the tables are created using JD Edwards EnterpriseOne standards.

17.2 Z Event Process Flow

This diagram shows Z event processing. The diagram expands on the system diagram provided in the Using Events - Guaranteed Overview chapter. This diagram details the processing that the CallObject kernel does during Z event processing. In the System Overview diagram, the BSFN uses the Event API, all within the CallObject kernel and in turn places the event data into the F90710 table. For Z events, additional processing occurs within the CallObject kernel before the event is placed into the F90701 table. Z events that are placed in the F90710 table are already in XML format (unlike real-time and XAPI events, which only have raw event data in the table).

Figure 17–1 Z event processing



In summary:

1. When a JD Edwards EnterpriseOne transaction occurs, the master business function writes the transaction information in the appropriate interface table and sends an update record to the F986113 table.
2. A batch process monitors the F986113 table. When the batch process finds a W status in the F986113 table, it notifies the Z Event Generator (ZEVG), which is part of the CallObject kernel. The batch process looks in the F0047 table to determine which Z-event generator to call.
3. The F47002 table provides a cross-reference between the transaction and the interface table where the record is stored. This information is used by the Z-event generator.
4. The Z-event generator retrieves the transaction information from the interface table and converts the transaction information into an XML document using a JD Edwards EnterpriseOne DTD.
5. The Z-event generator sends the event (in the form of an XML document) to the event API for distribution.

6. After an event is successfully generated, the successfully generated column in the F0046 table is updated. A UBE purges information in the interface table based on information in the F0046 table.
7. The Event API sends the XML document to the F90710 table, where it is retrieved by the Transaction server and routed to a subscriber.

17.3 Vendor-Specific Outbound Functions

The purpose of the vendor-specific outbound function is to pass the key fields for a record in the outbound interface tables to a third-party system. With these keys, you can process information from the database record into your third-party system. The generic outbound subsystem batch process calls the function.

Each vendor-specific function is specific to the transaction being processed. You must decide how the function actually uses the database record information. Although the functions are written to your specifications, and most likely are written outside of JD Edwards EnterpriseOne, these functions must use the required JD Edwards EnterpriseOne defined data structure:

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

17.4 Working With Z Events

This section provides an overview about Z event configuration and discusses how to add a data export control record.

17.4.1 Configuring Z Events

To generate Z events, complete these tasks:

- Enable the Z event.
- Update the Flat File Cross-Reference table.
- Update the Processing Log table.
- Verify the subsystem job is running.
- Purge data fro the interface table.
- Synchronize F47002 records with F90701 records.
- Set up data export controls.

17.4.2 Enabling Z Event Processing

You can enable or disable master business functions to write transaction information into interface tables and the F986113 table when a transaction occurs. All outbound master business functions that have the ability to create interoperability transactions have processing options that control how the transaction is written. On the Processing Options Interop tab, the first processing option is the transaction type for the interoperability transaction. If you leave this processing option blank, the system does not perform outbound interoperability processing. The second processing option controls whether the *before* image is written for a change transaction. If this processing option is set to 1, before and after images of the transaction are written to the interface table. If this processing option is not set, then only an *after* image is written to the interface table.

17.4.3 Updating Flat File Cross-Reference

When you enable Z events, you also update the F47002 table.The transaction type that you entered in the processing option maps to the F47002 table to determine in which interface tables to store the information from the transaction. You use the Flat File Cross-Reference program (P47002) to update the F47002 table.

17.4.4 Updating the Processing Log Table

The Z event generator uses the F0046 table. The F0046 table contains the keys to the interoperability transaction along with a successfully processed column. The sequence number, transaction type, order type, function name, and function library are obtained from the F0047 table. A vendor-specific record is sequentially created in the F0046 table for every transaction processed by the Interoperability Generic Outbound Subsystem (R00460) UBE or the Interoperability Generic Outbound Scheduler UBE (R00461). For example, if three vendors have subscribed to a transaction using the F0047 table, three records are created in the F0046 table, one record 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 can use the Processing Log (P0046) program to determine whether a vendor-specific object processed the interoperability transaction correctly.

A purging UBE that purges the interfaces tables runs based on information in the processing log table.

Data in the Processing Log table cannot be changed.

17.4.5 Verifying that the Subsystem Job is Running

When the application master business function adds a record to the F986113 table, a subsystem job is started. Subsystem jobs are continuous jobs that process records from the Subsystem Job Master table. You should verify that the subsystem job is running.

Note: After the records are processed, instead of ending the job, subsystem jobs look for new data in the data queue. Subsystem jobs run until you terminate them.

You can schedule subsystem jobs.

See "Understanding JD Edwards EnterpriseOne Subsystems" in the *JD Edwards EnterpriseOne Tools System Administration Guide*.

See "Understanding the Scheduler Application" in the *JD Edwards EnterpriseOne Tools System Administration Guide*.

17.4.6 Purging Data from the Interface Table

After you receive the Z event, you should purge the data from the interface table. You can enter a purge UBE in the Processing Log table to purge the interface table.

See Interoperability Interface Table Information.

See Purging Interface Table Information.

17.4.7 Synchronizing F47002 Records with F90701 Records

Z events that are automatically created write records to the F90701 table. If you have existing Z events defined and are upgrading to an 8.11 or later release, you can run the Populate Event Activation Status Table UBE (R90705)to create the associated F90701 table records for the pre-existing Z event definitions.

17.5 Setting Up Data Export Controls

This section provides an overview of setting up data export controls and discusses setting up the record.

17.5.1 Understanding Data Export Controls Records

The generation of outbound data is controlled through the F0047 table. You use the Data Export Controls program (P0047) to update the F0047 table. For each transaction type and order type, you must designate the Z event generator that will process the outbound data. To send a given transaction type to more than one third-party application, you associate the transaction type with each of the individual destinations by making separate entries in the F0047 table for each destination. JD Edwards suggests that you 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 that you can retrieve the transaction.

17.5.2 Forms Used to Add a Data Export Controls Record

Form Name	FormID	Navigation	Usage
Work with Data Export Controls	W0047A	From an application that supports event generation, open the Data Export Controls program An alternate way to access the Data Export Controls Program is to enter P0047 in the Fast Path command line	View existing data export control records.
Data Export Control Revisions	W0047C	On Work with Data Export Controls, click Add.	Add a new data export control record.

17.5.3 Adding a Data Export Control Record

Access the Data Export Control Revisions form.

To set up Data Export Controls:

1. Complete these fields:

  - Transaction
  - Order Type
2. For each detail row, enter one of these, depending on your platform:
  - Function Name

    Windows NT: _CallOnUpdate@36

    UNIX: CallOnUpdate

    IBM i: CallOnUpdate
  - Function Library

    Windows NT: EnterpriseOne Bin32 Path\zevg.dll

    UNIX(HP): EnterpriseOne Bin32 Path\libzevg.sl

    UNIX(AIX, SUN): EnterpriseOne Bin32 Path\libzevg.so

    IBM i: EnterpriseOne Bin32 Path\ZEVG
  - Enter 1 in the Execute For Add column to generate an event for an add or insert.

    Complete the same process as appropriate for update, delete, and inquiry.
  - 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.

    The system automatically increments the Sequence field for each line.

18

Using Batch Interfaces

This chapter contains the following topics:

18.1 JD Edwards EnterpriseOne Interface Tables

An interface table (also called a Z table) is a working table where non-JD Edwards EnterpriseOne information can be stored and then processed into JD Edwards EnterpriseOne. You can also use interface tables to retrieve JD Edwards EnterpriseOne data. JD Edwards EnterpriseOne interface tables mirror JD Edwards EnterpriseOne application tables.

JD Edwards EnterpriseOne provides predefined interface tables for some applications. You can also create your own interface tables as long as your interface table is formatted in accordance with JD Edwards EnterpriseOne standards.

If you receive an error message when the interface table is processed, you can use a revision application to make corrections to the data and then reprocess the data in batch or transaction mode. After you have successfully processed the data in the interface table, you should run a purge application to remove all records from the interface table and to any remove secondary interface tables from the system.

Note: You usually use a batch interface to collect transactions over a period of time and then process all of the transactions at once.

18.1.1 Structuring Interface Tables

Each JD Edwards EnterpriseOne transaction uses a set of interface tables. Some files share a common set of interface tables. The interface table name is based on the JD Edwards EnterpriseOne application table name and has Z1 as a suffix. For example, if the application table is the F4211 table, the interface table is the F4211Z1 table.

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

- Inbound is based on the application table that is updated with data from the interface table.

- Outbound is based on the application table that has data extracted from it and placed in the interface table.

Both the inbound and outbound directions of an internal transaction within a system use a single set of interface tables. For example, for a sales order in the Sales Order system, the inbound customer order (850) and the outbound order acknowledgment (855) share a set of interface tables.

If the interface table is used for both inbound and outbound transactions, the based-on table should be the same application table. In the Sales Order example with an inbound customer order and an outbound order acknowledgment, the detail interface table is based on the F4211 table.

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

The beginning of the table has these 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 previously discussed.

The end of the table has the these columns, which 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 interface 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 or match interface 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.

Note: If you create custom interface tables, use the structure and format described in this chapter.

18.1.2 Updating JD Edwards EnterpriseOne Records

You use interface tables to import non-JD Edwards EnterpriseOne transactions into the live JD Edwards EnterpriseOne database. These non-JD Edwards EnterpriseOne transactions are referred to as Z transactions. Inbound interface tables are based on the JD Edwards EnterpriseOne application table where the transaction is stored. Once records are correctly updated to the appropriate interface table, you can update the record to the JD Edwards EnterpriseOne database.

See Also:

- Understanding Z Transactions.

18.1.3 Retrieving JD Edwards EnterpriseOne Records

You can use interface tables to retrieve information from JD Edwards EnterpriseOne. Outbound interface tables are based on the JD Edwards EnterpriseOne application table from where the data is extracted. You can retrieve records from JD Edwards EnterpriseOne by running an extraction batch process, by using a subsystem business function, or by generating a Z event.

18.1.3.1 Running an Extraction Batch Process

You copy the records from the JD Edwards EnterpriseOne application tables to the JD Edwards EnterpriseOne outbound interface tables using the extraction batch process that is specifically set up for the type of document you are sending.

You initiate the extraction batch process for applications that support extraction batch processing. 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 JD Edwards EnterpriseOne application tables for the transaction and copies the data into the outbound interface tables. The system also generates an audit report that lists the records that completed successfully. Errors are placed on the audit report and also sent to the employee work center. You can use a revisions application to correct errors in the interface table records.

18.1.3.2 Subsystem Business Function

You can use the generic outbound subsystem business function, Add Transaction To Subsystem Queue (B0000176), to write a record to the subsystem data queue to specify a batch process that needs to be awakened in the subsystem. This 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 outbound transaction is:

- Line Number (EDLN)
- Transaction Type (TYTN)
- Document Type (DCTO)
- Action Code (TNAC)

See Also:

- Understanding Z Events - Guaranteed.
- Understanding Z Events - Classic.

18.1.4 Using the Revision Application

You use the revision application to add, delete, edit, and review transactions in the interface tables. You can use a revision application to correct the record in error. After you make a change to the interface table, you run the process again. You can continue to make corrections and rerun the transaction process until the program completes without errors. The name is based on the detail interface table. For example, if the tables for Sales Order Entry are F4201Z1 and F4211Z1, the revision application is P4211Z1. The revisions application can call the appropriate purge named event rule to delete records from the interface table.

18.1.5 Purging Interface Table Information

You should run a purge batch process periodically after you have successfully processed the data in the interface tables. The purge batch process should have one or two sections; the number of sections depends on the interface tables. The purge batch process calls the purge named event rule (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 NERs have two modes:

- Header mode, which deletes the header record and all associated records in 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 is 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.

18.2 Electronic Data Interface

The JD Edwards EnterpriseOne Data Interface for Electronic Data Interchange (EDI) system acts as an interface between the JD Edwards EnterpriseOne 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.

See Also:

- *JD Edwards EnterpriseOne Data Interface for Electronic Data Interchange 9.0 Implementation Guide*.

18.3 Table Conversion

Table conversion is a special form of Universal Batch Engine (UBE) that enables you to do high-speed manipulation of data in tables. JD Edwards EnterpriseOne has a table conversion utility that you can use to gather, format, export, and import enterprise data. The table conversion tool enables you to transfer and copy data and to delete records from tables.

See Also:

- "Understanding Table Conversion" in the *JD Edwards EnterpriseOne Tools Development Tools: Data Access Tools Guide*.

18.4 Output Stream Access UBEs

If you have set up an Output Stream Access (OSA) interface, you can pass JD Edwards EnterpriseOne data to another software program for processing and formatting. OSA can use its own set of commands or it can use an XML library.

See Also:

- "Understanding OSA" in the *JD Edwards EnterpriseOne Tools Development Tools: Report Printing Administration Technologies Guide*.

18.5 Advanced Planning Agent Integration

The JD Edwards EnterpriseOne Advanced Planning Agent (APAg) is a tool for batch extracting, transforming, and loading of data. APAg supports access to data sources in the form of relational databases, flat file format, and other data or message encoding, such as XML. APAg also moves data from one place to another and initiates tasks related to the movement of the data.

See *JD Edwards Supply Chain Planning, Advanced Planning Agent Guide*.

19

Using Open Data Access

This chapter contains the following topics:

19.1 Understanding Open Data Access

The JD Edwards EnterpriseOne Open Data Access ODBC driver is a read-only driver that is compliant with version 2.5 or higher. Front-end Windows query and reporting tools can use ODA to access the JD Edwards EnterpriseOne database. ODA supports these front-end tools:

- Microsoft Query
- Microsoft Access
- Microsoft Excel
- ODBCTEST
- Crystal Reports
- Microsoft Analysis Service (not certified)

ODA sits between the front-end Query and Reporting tool and the JD Edwards EnterpriseOne-configured ODBC drivers.

19.2 Installing ODA

To access JD Edwards EnterpriseOne data with the ODA ODBC driver, your system must meet the minimum technical requirements (MTR) for JD Edwards EnterpriseOne. MTRs are updated for each release and are available on the My Oracle Support website. Before you install ODA, ensure that your system meets the specified hardware and software requirements.

19.2.1 Hardware Requirements

Hardware requirements include:

- IBM-compatible personal computer.

- Hard disk with 6 MB of free disk space.
- At least 16 MB of random access memory (RAM).

19.2.2 Software Requirements

Software requirements include:

- JD Edwards EnterpriseOne.
- JD Edwards EnterpriseOne Open Data Access driver (JDEOWODA.dll).
- The 32-bit ODBC Driver Manager, version 3.0 or later (ODBC32.dll).

  This file is included with the ODBC Database Drivers.
- Microsoft Windows 95 or later, or Windows NT 4.0 or later.

> **Note:** The use of the ODA ODBC driver by 16-bit applications on Windows 95 is not supported.

19.2.3 ODBC Component Files

The JD Edwards EnterpriseOne installation installs the components required by ODBC database drivers. You might also find these additional files:

File	File Name
ODA Driver	JDEOWODA.DLL
ODA Driver Help	JDEOWODA.HLP
Release Notes	README.TXT

> **Note:** OLEDB is a driver for SQL Server. However, OLEDB data source is not supported for ODA. If you are using ODA with SQL Server, use ODBC to set up your data source.

19.2.4 ODA Driver Architecture

The JD Edwards EnterpriseOne ODA ODBC driver architecture has five components:

Component	Description
Application	A front-end Query and Reporting tool that calls the ODA driver to access data from the JD Edwards EnterpriseOne database.
Manager	Loads and unloads drivers on behalf of an application. Processes ODBC calls or passes them to the ODA driver.
JD Edwards EnterpriseOne ODA Driver	Passes some of the ODBC requests directly to the vendor's ODBC driver. If specific data types for JD Edwards EnterpriseOne 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 JD Edwards EnterpriseOne ODA ODBC driver might need to manipulate the data so that it displays 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.

Component	Description
Data Source	The data that you want to access, as well as the operating system, DBMS, and network platform for the data.

19.3 Working with Data Sources

This section provides an overview of data sources and discusses how to

- Add a data source.
- Modify a data source.
- Delete a data source.
- Configure a data source.
- Connect a data source.

Although the ODA driver is automatically registered as part of the installation process, you might need to add a driver data source. You can also add a file data source or a system data source. A system data source can be used by more than one user on the same machine. A system data source is a data source that you have set up with a system data source name (DSN). The system DSN can also be used by a system-wide service, which can then gain access to the data source even if no user is logged on to the machine. You can delete any of the data sources.

After you add a data source, you must configure and connect it. You can modify the configuration and connection setting for an existing data source. For example, you can configure the ODA driver so that you can view currency data in the correct format.

If you use Oracle, you must create another ODBC DSN, named OneWorld ODA Ora, so that you can access the Oracle data source through ODA. Specific information for doing this is included in the online release notes.

You can customize the list of functions that are enabled in ODA. Advanced configuration is optional. If you choose not to customize the list of functions enabled in ODA, the system uses a default list of settings.

You access the ODBC button from the Control Panel on your Windows workstation. When you click the ODBC button, a User Data Sources dialog box appears.

19.3.1 Adding a Data Source

After you add the data source, you must configure it and connect it. This table explains how to navigate on the User Data Sources dialog box to add a data source:

Function	Navigation on User Data Sources dialog box
Add an ODA Driver Data Structure	On the User Data Sources dialog box, click Add. On Add Data Source, select the JD Edwards EnterpriseOne Open Data Access driver from the Installed ODBC Drivers list, and then click Finish.
Add a File Data Source	On the User Data Sources dialog box, click the DSN tab. On File Data Sources, click Add. On Add Data Source, select the JD Edwards EnterpriseOne Open Data Access driver from the Installed ODBC Drivers list, and then click Finish.

Function	Navigation on User Data Sources dialog box
Add a System Data Source	On the User Data Sources dialog box, click the System DSN tab, and then click Add. On system Data Sources, click Add. On Add Data Source, select the JD Edwards EnterpriseOne Open Data Access driver from the Installed ODBC Drivers list, and then click Finish.

19.3.2 Modifying a Data Source

You can modify an existing data source. After you access the appropriate data source, select Configure and then modify the existing configuration settings.

19.3.3 Deleting a Data Source

To delete a data source, access the appropriate data source, select remove, and click Yes to confirm the delete.

19.3.4 Configuring a Data Source

To modify an existing data source, access the appropriate data source type and then select a data source from the available list. Click Configure. When you add a data structure, the Configure Data Source tab appears. Enter the information as shown in this table, and then click OK:

Field Name	Description
Data Source Name	Specify the name for the JD Edwards EnterpriseOne Open Data Access driver.
Description	Specify the description of the driver that you are adding. The Description entry cannot exceed 79 characters.

19.3.5 Connecting a Data Source

After the data source is configured, the Connect form appears. You can also select one or more table and business view display Options. On the Connect form, select one or more of these options:

Option Name	Description
Convert User Defined Codes	Select 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 option is to display the associated description instead of the user-defined code.
Convert Currency Values	Select this option to convert currency fields to the correct values.
Use Long Table or Business View Names	Select this option to view long table or view names.
Use Long Column Names	Select this option to view long column names
Tables Only	Select this option to view only JD Edwards EnterpriseOne tables.
Business Views Only	Select this option to view only JD Edwards EnterpriseOne business views.

Option Name	Description
Tables and Business Views	Select this option to view both JD Edwards EnterpriseOne tables and JD Edwards EnterpriseOne business views.

19.4 Working with ODA

This section discusses how to:

- Manipulate data.
- Use keywords in the connection string.
- Run a query using Microsoft Excel.

19.4.1 Manipulating Data

The JD Edwards EnterpriseOne 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, media object, 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 chosen tool. Once the ODA driver is properly installed and an ODBC data source is established, you can use the functionality of the ODA driver. When a SQL connection is established, the environment of the current connection is stored in the system as the database name. SQLGetInfo can access this value later or it can be used for future connections.

You can use these specific JD Edwards EnterpriseOne features with JD Edwards EnterpriseOne ODA:

Feature	Description
Long Table and Business View Names	Long table and business view names enable you to see a descriptive name when you view an object list. You can use either the descriptive names or the original JD Edwards EnterpriseOne object name in the SELECT statement. **Note:** This option might 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 enable you to see a descriptive name when viewing any columns list. You can still use either the descriptive names or the original JD Edwards EnterpriseOne column name. For example, you can use either of these statements to retrieve information from the F0101 table: ■ SELECT ABAN8 from the F0101 table. ■ SELECT AddressNumber from the F0101 table.

Feature	Description
Julian Date	Julian date modifies all references to Julian date columns to convert the date to an SQL-92 standard date. The JD Edwards EnterpriseOne Julian date is converted to a standard date value that can be used in date calculations. This feature enables you to use duration or other date calculations in both the SELECT (result data), WHERE, and HAVING clauses and the ORDER BY clause. The SQL SELECT statement is modified to before a data calculation to convert the JD Edwards EnterpriseOne Julian date column to a standard date. The modification to the SQL SELECT statement is based on 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 results in a date value of 1899-12-31. To remove these values, this condition should be added to the WHERE clause in the SELECT statement, where DATECOL is the JD Edwards EnterpriseOne 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 feature enables SQL statements that contain 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 JD Edwards EnterpriseOne 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. Before selected columns are returned, the JD Edwards EnterpriseOne 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 non-converted currency value. Currency columns in the GROUP BY or ORDER BY clause are grouped and sorted by the non-converted currency value.
Media Object	The Media object column, TXVC, in the F00165 table storage is limited to single-column references in the selected columns list. ODA returns media data in plain text or rich text format (RTF) and truncates other binary data, such as an image. The size limitation of the text or RTF is 30,000 characters, and text will be truncated when it reaches this limitation.
Column Security	When column security is active, any reference to restricted columns causes an error to be returned when the SELECT statement is examined, including the use of * (asterisk-selecting all columns) in the select clause, as defined by the SQL-92 standards. You will receive an error if you are not authorized for all of the columns in the table.
Row Security	When row security is active, the statement is modified to include the appropriate WHERE clause for filtering secured rows. You will only see rows that you are authorized to access along with getting accurate results using aggregate functions-for example, SUM or AVG.

Feature	Description
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 and so on). This is an optional setting that can be configured when you set up the driver. Before the UDC is returned to you, the JD Edwards EnterpriseOne 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 non-converted code and the UDC columns referenced in the GROUP BY and ORDER BY clause are grouped and sorted by the non-converted code.

19.4.2 Using Keywords in the Connection String

This section discusses keywords that you can use in a connection string when you write your own database applications.

You can use C programming language to write database applications that directly invoke SQL APIs that are supported by ODA, such as SQLDriverConnect and SQLBrowseConnect. This table lists keywords that you use in the connection string when you write your own database applications:

Key	Value	Description	Input Connection String	Output Connection String
CONVERTUDC	Y or N (default value is N)	Convert UDC or not	Optional. If not in the connection string, load from INI/registry settings (JD Edwards EnterpriseOne ODA DSN settings).	From the input string or INI/registry settings.
CONVERT CURRENCY	Y or N (default value is N)	Convert currency or not	Optional. If not in the connection string, load from INI/registry settings (JD Edwards EnterpriseOne ODA DSN settings).	From the input string or INI/registry settings.
SHIFTDECIMAL S	Y or N (default value is Y)	Use decimal shift or not	Optional. If not in the connection string, load from INI/registry settings (JD Edwards EnterpriseOne ODA DSN settings).	From the input string or INI/registry settings.

Key	Value	Description	Input Connection String	Output Connection String
CONVERTJULIAN DATES	Y or N (default value is Y)	Convert Julian dates or not	Optional. If not in the connection string, load from INI/registry settings (JD Edwards EnterpriseOne ODA DSN settings).	From the input string or INI/registry settings.
DISPLAYOPTIONS	0/1/2 (no default value)	Display TBLE, BSFN or both	Optional. If not in the connection string, load from INI/registry settings (JD Edwards EnterpriseOne ODA DSN settings).	From the input string or INI/registry settings.
LONGTABLE NAMES	Y or N (default value is Y)	Use long names for tables or not	Optional. If not in the connection string, load from INI/registry settings (JD Edwards EnterpriseOne ODA DSN settings).	From the input string or INI/registry settings.
LONGCOLUMN NAMES	Y or N (default value is Y)	Use long names for columns or not	Optional. If not in the connection string, load from INI/registry settings (JD Edwards EnterpriseOne ODA DSN settings).	From the input string or INI/registry settings.
UID	<string>	User ID	Required by JDEDriverConnect (SQL_DRIVER_NOPROMPT).	The same as the input if not overwritten by OW login.
PWD	<string>	Password	Required by JDEDriverConnect (SQL_DRIVER_NOPROMPT).	The same as the input if not overwritten by OW login.
ENVIRONMENT	<string>	Environment	Required by JDEDriverConnect (SQL_DRIVER_NOPROMPT).	The same as the input if not overwritten by OW login.
DBQ	<string>	The same as the ENVIRONMENT	Work as ENVIRONMENT, if ENVIRONMENT not specified.	Removed if ENVIRONMENT exists.

Key	Value	Description	Input Connection String	Output Connection String
DSN	<string>	Data source	Optional. Uses DEFAULT if invalid.	Overwritten by login.

If you use the Microsoft Analysis Service tool, you can use connection string keywords to create a new data source. Use this example to write a connection string:

```
DSN=OneWorld ODA;DBQ=ADEVHP02;
```

19.4.3 Running a Query Using Microsoft Excel

This section discusses how to use Microsoft Excel to create and run a query.

To run a query using Microsoft Excel:

1. From the Data menu, select Get External Data.
2. Select Create New Query.
3. On the Databases tab, select the appropriate data source (for example, JD Edwards EnterpriseOne Local or JD Edwards EnterpriseOne ODA).

   Because Excel uses file data sources, the ODA data source you set up in the 32-bit ODBC Administrator does not appear on the list of databases. You should create a File-type Data Source by selecting *New Data Source* and then follow the procedures for setting up a data source.

   When you select the ODA data source, you might need to log on to JD Edwards EnterpriseOne to use the ODA driver. Once you log on, you will not see the Solution Explorer because it is only activated so that the ODA driver can check security and environment mappings.

   The Excel Query Wizard displays a list of available tables in the JD Edwards EnterpriseOne data source. Expanding any table name shows the available columns or fields in each table. If you are using the ODA driver, you see long descriptions of each field (for example, DateUpdated). If not, you see the alpha codes for the fields (for example ABUPMJ).
4. To translate field and column names from the JD Edwards EnterpriseOne alpha codes, use the F9202 table. Select all rows and sort (on FRDTAI) to create a cross-reference.

   The first two letters of all JD Edwards EnterpriseOne 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.
6. Run your query and review it in Excel or MicroSoft Query.

   After you run a query from Excel, if you view the results using Microsoft Query, results are returned quickly. MicroSoft Query selects a page at a time. If you are working with a large result set, you should close JD Edwards EnterpriseOne and any applications that require a lot of memory so that you can more quickly navigate through the records. If you convert the query results directly into a spreadsheet instead of into Microsoft Query, the process might take significantly longer, and you cannot view the results until the entire file builds.

To verify the outcome of each query, you should run each one first using the non-ODA JD Edwards EnterpriseOne data source and then use the ODA data source and compare the results.

19.5 Managing ODA Error Messages

This section discusses error messages that you might receive.

JD Edwards EnterpriseOne Open Data Access driver sends error messages. The messages are placed in the ODBC error message queue where the application can retrieve them using the standard ODBC error mechanism. The JD Edwards EnterpriseOne messages look like this:

```
[J.D. Edwards][OneWorldODA Driver]MESSAGE TEXT
```

This is a list of the errors that you can receive from the driver:

Error Message	Description
Configuration Request Error	This error might 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 value and uses this message to let you know that the value has changed. The JD Edwards EnterpriseOne Open Data Access driver changes values in these areas: Setting the row set size to a value other than one. The driver currently only supports single-row row sets. Setting the login time out to a value other than zero. The driver currently only supports zero in this option, which means, timeout disabled.
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 might need to create a default data source.
Unable to Allocate Memory	The JD Edwards EnterpriseOne Open Data Access driver was not able to allocate enough memory to continue. You must close some applications and try the operation again. Make sure that you meet the minimum system requirements.

Error Message	Description
Invalid Type of Request	You attempted to use a configuration option that is unknown to the driver. The driver supports these 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 JD Edwards EnterpriseOne environment. The JD Edwards EnterpriseOne Open Data Access driver currently supports tables that are 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 JD Edwards EnterpriseOne Environment	The driver could not establish a connection to the JD Edwards EnterpriseOne 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 JD Edwards Enterprise 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 JD Edwards EnterpriseOne Open Data Access driver only allows such columns to be simple references.
Media Object Columns Can Only Be in Simple Column References	A user attempted to use a Media Object column in a complex expression. The JD Edwards EnterpriseOne 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.

Error Message	Description
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 not supported by the JD Edwards EnterpriseOne Open Data Access driver.
Invalid Date or 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 select 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 select 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 JD Edwards EnterpriseOne 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 row set 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 JD Edwards EnterpriseOne Open Data Access driver substituted a similar value.

Error Message	Description
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 select 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 JD Edwards EnterpriseOne Open Data Access driver restricts the SELECT statement to contain only one business view.
Unable to Open Table or Business View	The JD Edwards EnterpriseOne Open Data Access driver was unable to locate the table or business view in the JD Edwards EnterpriseOne database or could not get information pertaining to the table or business view.
Server Connection Failed	The JD Edwards EnterpriseOne 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 JD Edwards EnterpriseOne Open Data Access driver.
Business View Contains Unsupported UNION Operator	The Business View definition contains the UNION operator, which could not be processed by the JD Edwards EnterpriseOne Open Data Access driver.

20

Using the Java Database Connectivity Driver

This chapter contains the following topics:

- Section 20.1, "Using the JDBC Driver"
- Section 20.2, "JDBC Driver Configuration"
- Section 20.3, "JDBC Driver Connection Details"
- Section 20.4, "JDBC Driver Security Considerations"
- Section 20.5, "SQL Grammar"
- Section 20.6, "JDBC Driver Features"
- Section 20.7, "JDBC Driver Troubleshooting"
- Section 20.8, "JDBC Driver Terminology"

20.1 Using the JDBC Driver

A JDBC driver is a software component that enables a Java application to interact with a database. Four types of JDBC drivers are available. Oracle JD Edwards EnterpriseOne supports Type 3 and Type 4 JDBC drivers.

This table provides an overview of each of the four types of JDBC drivers:

JDBC Driver Type	Description
Type 1 JDBC driver	Type 1 JDBC drivers translate JDBC calls into ODBC calls. Type 1 JDBC drivers are usually called JDBC-ODBC bridge drivers.
Type 2 JDBC driver	Type 2 JDBC drivers translate JDBC calls into native DBMS APIs. The Type 2 drivers consist of a Java component and a native code component, which requires that binary code be loaded on each client machine.
Type 3 JDBC driver	Type 3 JDBC drivers are pure Java drivers that use database middleware. The Type 3 drivers communicate with the database through middleware servers that must be running in the network. The net protocol allows the client JDBC drivers to be very small and to load quickly. Fetching data rows may take longer because the data comes through a middleware server. The JD Edwards EnterpriseOne Data Access Server (DAS) is a read-only Type 3 JDBC driver. The client is a small jar file that requires no configuration. The driver accesses the database through a DAS server. The DAS server is administered through Server Manager.

JDBC Driver Type	Description
Type 4 JDBC driver	Type 4 JDBC drivers are pure Java drivers that access a database directly. The Type 4 drivers are sometimes called thin drivers. Type 4 JDBC drivers have relatively fast performance. The JD Edwards EnterpriseOne Data Access Driver (DADriver) is a read-only type 4 JDBC driver. The DADriver client consists of many jar files and configuration files. The installation and administration is facilitated by Server Manager

The JD Edwards EnterpriseOne JDBC drivers provide read-only access to JD Edwards EnterpriseOne application and product data. In addition to masking the details for the many supported databases and platforms that JD Edwards EnterpriseOne products support, the JDBC drivers encapsulate additional filtering and processing that must occur in order to preserve data and semantic integrity.

The JD Edwards EnterpriseOne JDBC drivers provide Java applications with a logical connection to JD Edwards EnterpriseOne data. Applications view this logical connection as a normal database connection, despite the fact that specific data source details are hidden. In some cases, the JDBC driver maps a single logical connection to multiple physical data sources. In a sense, the JDBC driver presents the set of data that JD Edwards EnterpriseOne products manage as a database.

20.1.1 When to Use a JDBC Driver

Use a JD Edwards EnterpriseOne JDBC driver if you are developing or using software that requires or expects you to plug in a JDBC driver for data access, and you need to interact with JD Edwards EnterpriseOne application and product data.

20.1.1.1 Prerequisites

If you are using a Type 3 JDBC driver, you must install JD Edwards EnterpriseOne Tools Release 8.98 or later Tools software version.

If you are using a Type 4 JDBC driver, you must install JD Edwards EnterpriseOne Tools Release 8.98.1 or later Tools software version.

20.1.1.2 Using the Type 3 JDBC Driver

If you are trying to read small amounts of data using an interoperability client over a network, use the Type 3 JDBC driver. This list provides some examples for using the Type 3 JDBC driver:

- When using a commercial database middleware library (such as TopLink).
- When using a commercial database visualization tool (such as DBVisualizer).
- When retrieving JD Edwards EnterpriseOne data into a spreadsheet that has JDBC features (such as Excel).

The Type 3 driver can support approximately 1,000 desktops.

20.1.1.3 Using the Type 4 JDBC Driver

If you are trying to read large amounts of data, use the Type 4 JDBC driver. This list provides some examples for using the Type 4 JDBC driver:

- When using the Oracle BI Publisher Enterprise Edition reporting tool.
- When using any other commercial reporting tool.

20.1.2 Connection Mode

The JD Edwards EnterpriseOne product suite employs a diverse set of data sources. Specific filtering must occur for certain data sources while others can be used as is. The JD Edwards EnterpriseOne JDBC drivers define various connection modes that indicate the type of additional filtering and processing that the data requires. Application code designates the connection mode when it establishes new connections.

Currently the only connection mode supported is *enterpriseone*, which establishes a connection for reading JD Edwards EnterpriseOne enterprise resource planning (ERP) 9.0 data. This connection mode is implemented using JDBj, the Java class library that encapsulates most aspects of ERP data access middleware functionality such as object configuration management (OCM), ERP triggers, ERP business views, ERP row and column security, and decimal scrubbing.

The *enterpriseone* connection mode provides read-only access to ERP data. The concept of connection modes enables the extension of the JD Edwards EnterpriseOne JDBC drivers for other JD Edwards EnterpriseOne products as well.

20.2 JDBC Driver Configuration

Server Manager installs the components for both of the JD Edwards EnterpriseOne JDBC drivers.

See "Create a JD Edwards EnterpriseOne Data Access Server as a New Managed Instance" in the *JD Edwards EnterpriseOne Tools Server Manager Guide* on My Oracle Support.

See "Install a JD Edwards EnterpriseOne Data Access Driver" in the *JD Edwards EnterpriseOne Tools Server Manager Guide* on My Oracle Support.

Important: If you are using a Type 3 JDBC driver, you must configure the JDBC driver by copying the *e1jdbc.jar* driver jar file to the class path of the application that will use the JDBC driver. The *e1jdbc.jar* jar file is located in the classes folder of the JD Edwards EnterpriseOne Data Access Server (DAS).

The Type 4 JDBC driver does not require manual configuration.

20.3 JDBC Driver Connection Details

Java code that uses a JDBC driver must register the driver class name and designate a connection URL and optional connection properties that collectively identify the data source that the JDBC driver is accessing.

20.3.1 Driver Class Name

You must register the JD Edwards EnterpriseOne JDBC driver class name with the JDBC Driver Manager before attempting to use the driver. You register the JD Edwards EnterpriseOne JDBC driver using Class.forName. The following table shows the Type 3 and Type 4 JDBC driver class names.

JDBC Driver	Class Name
Type 3 JDBC Driver	com.jdedwards.jdbc.driver.Driver

JDBC Driver	Class Name
Type 4 JDBC Driver	com.jdedwards.jdbc.driver.JDBCDriver

The following table provides example registrations for the Type 3 and Type 4 JDBC drivers:

JDBC Driver Type	Example Registration
Type 3 JDBC Driver	Class.forName("com.jdedwards.jdbc.driver.Driver")
Type 4 JDBC Driver	Class.forName("com.jdedwards.jdbc.driver.JDBCDriver")

Some environments provide alternate mechanisms for registering JDBC drivers.

20.3.2 Connection URL

You must pass the following values to DriverManager.getConnection when establishing a JD Edwards EnterpriseOne JDBC connection:

- Connection mode: enterpriseone.
- Connection target: The ERP environment.
- User name and password: The ERP user name and password.

The connection mode designates the type of JD Edwards EnterpriseOne product data that you plan to access.

The connection target, user name, and password depend on the connection mode.

The format for the connection URL is:

```
jdbc:oracle:connectionMode://<environment>
```

> **Note:** If you are using the Type 3 JDBC driver, include the host name and port number of the DAS server, for example:`jdbc:oracle:connectionMode://hostname:port/<environment>`

20.3.3 Connection Properties

The JD Edwards EnterpriseOne JDBC drivers recognize several connection properties that you can set when you establish a new connection. You specify these in the connection URL or in the java.util.Properties object that you pass to DriverManager.getConnection. If you specify the same property in both places, the value in the URL takes precedence.

If the property value contains one or more semicolons, you may need use parentheses to delimit the property value. Otherwise, parentheses are optional.

The following table shows the connection properties that the JD Edwards EnterpriseOne JDBC drivers recognize. The set of valid connection properties varies based on the connection mode. The JD Edwards EnterpriseOne JDBC drivers ignore any connection properties that are not listed in this table:

Connection Mode	Property Name	Property Value
enterpriseone	enterpriseone.role	The ERP role, if any. The default is *ALL. This property value applies only if you are accessing ERP 9 or later data.
enterpriseone	impersonate	The user name, which will be substituted for authorization purposes at runtime in a proxy authentication mode. This is discussed in the JDBC Security Considerations section.

20.3.3.1 Example Showing How to Use Connection Properties

This example code shows how to connect to ERP environment ADEVHPO2I:

```
Connection connection
   = DriverManager.getConnection(
   "jdbc:oracle:enterpriseone://ADEVHPO2I;",
   "myuser",
   "mypassword");
```

Note: If you are using a Type 3 driver, include the host name and port of the DAS server, for example: “jdbc:oracle:enterpriseone://hostname:port/ADEVHPO2I;”,

---

20.4 JDBC Driver Security Considerations

JD Edwards EnterpriseOne JDBC drivers require a user name and password for authentication. At the same time, the same user name is authorized for the environment and role, which are passed in the connection URL. If you do not specify a role in the connection URL, the system uses *ALL. This model poses a serious security risk and a high maintenance requirement for third-party systems where a single JDBC connection is shared across multiple users.

To alleviate this problem, the JD Edwards EnterpriseOne JDBC drivers allow for a proxy authentication model by way of the *impersonate* connection property. In this model, the authentication and authorization are separated into two steps:

1. All users are authenticated through the security server with a sign-on EnterpriseOne proxy user name and password.

   Important: If you are using a Type 3 JDBC driver, this user name *must* be the same as the JDBj Bootstrap session user ID of the Data Access Server instance to which you are connecting.

   ---

2. The *impersonate* user name that is passed in the connection property, is authorized for the environment and role. If you do not specify a role in the connection URL, the system uses *ALL.

20.5 SQL Grammar

The JD Edwards EnterpriseOne JDBC drivers support different flavors of SQL depending on the connection mode.

20.5.1 SQL Grammar for JD Edwards EnterpriseOne Connection Modes

The JDBC drivers implement JD Edwards EnterpriseOne connection modes using JDBj, which is a Java data access API. The JDBC drivers parse SQL statements and transforms them into JDBj operations.

In general, the JDBC driver using the EnterpriseOne connection accepts only SELECT statements. All other operations, such as INSERT, UPDATE, DELETE, ALTER, DROP, and CREATE statements are not supported. If the driver cannot parse the SQL statement, then the JDBC driver throws an SQLException with a message that explains the parsing error.

The following table describes the SQL grammar that the parser recognizes. In this table, SQL keywords are in bold font (**SELECT**.) SQL keywords are not case sensitive. Rule names are listed in italics (*where-clause*.) Terminal symbols are noted. Optional clauses are listed in square brackets (,[order-by-clause].) Clauses that may repeat 0 or more times are listed in parenthesis followed by an asterisk ((, database-object-with-alias)*.) A vertical bar indicates that one of a set of options is valid (* | fields).

Rule	Definition
select-statement	**SELECT** *fields-clause* **FROM** *database-objects [where-clause] [group-by-clause] [order-by-clause]*
subquery-clause	**SELECT** *fields-clause* **FROM** *database-object-with-alias [where-clause] [group-by-clause]*
database-objects	*database-object-with-alias* (, *database-object-with-alias*)*
database-object-with-alias	*database-object [ID]* **Note:** ID is a terminal symbol.
database-object	*ID* **Note:** ID is a terminal symbol. Database object names are table and business view names. Do not qualify these with an owner or schema. The JDBC driver uses its own data source resolution mechanisms (such as an ERP system's OCM) to resolve database object name qualifiers. However, if you require a schema to satisfy some third-party software requirements, you qualify the table or business view names with *JDE* as the schema. JD Edwards EnterpriseOne does not have a schema or catalog concept and this qualification is ignored at runtime.
fields-clause	* | *fields* | *field-function-expressions*
fields	*field* (, field)* | *field* **AS** *alias* (, *field* **AS** *alias*)*
field	*database-object* [. ID [. *field-instance*]] **Note:** ID is a terminal symbol. Field names are in the format database-object.field.instance, where database-object and instance are optional. Field names match data dictionary names rather than physical column names. For example, use AN8 (the data dictionary name for address book number) rather than ABAN8 (the physical F0101 column name). Instance is an integer that refers to the instance of a particular field when used in a self-join.
field-instance	INTEGER_LITERAL **Note:** INTEGER_LITERAL is a terminal symbol.
field-function-expressions	*field-function-expression* (, *field-function-expression*)* | *field-function-expression* **AS** *alias*(, *field-function-expression* **AS** *alias*)*

Rule	Definition
field-function-expression	*type1-field-function-expression* | *type2-field-function-expression* | *type3-field-function-expression*
type1-field-function-expression	**AVG | COUNT | SUM[DISTINCT]** (*field*) **Note:** See the examples provided in the following table.
type2-field-function-expression	**MIN | MAX** (*field*) **Note:** See the examples provided in the following table.
type3-field-function-expression	**COUNT (*)**
field-reference	*field*
literals	*literal (, literals)**
literal	STRING_LITERAL | INTEGER_LITERAL | FLOATING_POINT_LITERAL | **NULL** | **?** **Note:** STRING_LITERAL, INTEGER_LITERAL, and FLOATING_POINT_LITERAL are terminal symbols.
where-clause	**WHERE** *or-expression*
group-by-clause	**GROUP BY** *group-by-fields*
order-by-clause	**ORDER BY** *order-by-fields*
order-by-fields	*order-by-field-and-direction*(, *order-by-field-and-direction*)*
order-by-field-and-direction	*field-reference* [*order-by-direction*]
order-by-direction	**ASC | DESC**
or-expression	*and-expression* (**OR** *and-expression*)*
and-expression	*not-expression* (**AND** *not-expression*)*
not-expression	[**NOT**] *sub-expression*
sub-expression	*exists-clause* | *relational-expression* | **(***or-expression***)**
exists-clause	**EXISTS** *(subquery-clause)*
relational-expression	*field field-expression* | *in-expression* | *between-expression* | *like-expression* | *is-null-expression* **Note:** Inconsistent results might occur if you use a field that requires decimal scrubbing within a relational expression.
field-expression	*comparison-op* (([**ALL** | **ANY**] **(** subquery-clause **)**) | *element*)
in-expression	[**NOT**] **IN(***subquery-clause* | *elements* **)**
between-expression	[**NOT**] **BETWEEN***element* **AND** *element*
like-expression	**LIKE** *element*
is-null-expression	**IS** [**NOT**] **NULL**
elements	*element* (, *element*)*

<table>
<tr><th>Rule</th><th>Definition</th></tr>
<tr><td>element</td><td>field-reference | literal</td></tr>
<tr><td>comparison-op</td><td>= | != | <> | > | >= | < | <= | *= | =* | *=*</td></tr>
</table>

The following are some examples of SQL statements that are allowed:

Object Type	Statement

Table	`select AN8 from F0101`
	`or`
	`select AN8 AS AddressBookNumber from F0101`

Select All Table	`select * from F0101`
Table Join	`select avg(t1.an8), min(t1.an8),max(t1.an8), count (t1.Name), sum(t1.an8), avg(distinct t1.an8), count (distinct t1.name),sum(distinct t1.an8),t1.an8 from F0101 to, F0010 t1 where t0.an8=t1.an8 group by t1.an8`
Table Union	`select F4211.KCOO, F4211.DOCO, F4211.DCTO , MAX (F4211.LNID), COUNT(F4211.DOCO), MIN(F4211.LNID), min (F4211.AN8) from F4211 group by F4211.LNID,F4211.DOCO, F4211.DCTO,F4211.KCOO UNION select F42119.KCOO, F42119.DOCO, F42119.DCTO , MAX(F42119.LNID), COUNT (F42119.DOCO), MIN(F42119.LNID), min(F42119.AN8) from F42119 group by F42119.LNID, F42119.DOCO, F42119.DCTO,F42119.KCOO order by F4211.DOCO DESC, F4211.KCOO asc`
Single Table Business View	`select AN8 from V0101C`
Multiple Table Business View	`select F0101.AN8, F0116.AN8 from V0101JE`
Union Business View	`select max(F4211.KCOO), max(F4211.KCOO) from V4211A`

20.6 JDBC Driver Features

The JD Edwards EnterpriseOne JDBC drivers support different JDBC features depending on the connection mode. In general, the JDBC drivers implement the JDBC 3.0 specification as it is defined in Java 2 Platform Standard Edition version 5.0 (also called version 1.5.)

20.6.1 JDBC Features for the Connection Mode

The JDBC driver *enterpriseone* connection mode explicitly does *not* support the following JDBC features:

- Catalog methods (in DatabaseMetaData) with the exception of getCatalogs, getSchemas, getTables and getColumns.
- Cursor names (Statement.setCursorName and ResultSet.getCursorName).
- ResultSetMetaData as returned by PreparedStatement.getMetaData (the same information is available from ResultSet.getMetaData).
- Result set holdability (Connection.createStatement, Connection.prepareStatement, Connection.prepareCall, and Statement.getResultSetHoldability).
- Savepoints (Connection.setSavepoint and Connection.rollback).

- Scrollable result sets (Connection.createStatement, Connection.prepareStatement, Connection.prepareCall, Statement.getResultSetType, and ResultSet.getType).
- Stored procedures (Connection.prepareCall).
- Type map (Connection.setTypeMap, Connection.getTypeMap, and ResultSet.getObject).
- Update operations that involve JD Edwards EnterpriseOne software data (Statement.executeUpdate, PreparedStatement.executeUpdate, and ResultSet update methods).

In most cases, invoking these features results in an SQLException with a message describing the specific feature that is not supported.

20.7 JDBC Driver Troubleshooting

When errors occur, the JDBC driver throws SQLExceptions. In your code, it is helpful to print or log these exceptions so that you can inspect or report them as part of the troubleshooting process. It is especially helpful to inspect entire exception stack traces, because traces include exception messages, class names, lines numbers, and cause exceptions that lead to SQLExceptions.

When you evaluate a series of exceptions in a trace, you should concentrate on the first exception because it is often the cause of subsequent exceptions.

Some example exceptions and their recovery are discussed here.

20.7.1 No Suitable Driver

Exception: java.sql.SQLException: No suitable driver

Cause: The JD Edwards EnterpriseOne JDBC drivers use the native database JDBC drivers to access physical data. If the class path does not include the necessary drivers, then the JDBC drivers throw this exception on any attempt to read physical data.

Recovery: For the Type 3 JDBC driver, contact your system administrator and ensure that all of the applicable JDBC drivers are included in the Data Access Server's class path.

For the Type 4 JDBC driver, contact your system administrator and ensure that all of the applicable JDBC drivers are included in the same class path as the Data Access Driver.

20.7.2 Data Source for F0010, TBLE Not Found

Exception: com.jdedwards.services.objectlookup.DataSourceNotFoundException: Data source for F0010, TBLE not found. (with a cause message in parenthesis)

Cause: This exception indicates that the JDBC driver cannot access its system tables in ERP mode. Table F0010 is the first system table that the JDBC driver attempts to access. Be sure to check the cause message that is attached to the exception message. The exception trace usually includes a direct cause as well.

Recovery: Check the cause exception and follow the recovery instructions listed for those exceptions. If none apply, contact your system administrator and verify that the [JDBj-BOOTSTRAP DATA SOURCE] section of jdbj.ini file references a valid data source. The JDBj-BOOTSTRAP DATA SOURCE section describes the location for the ERP system tables, like F0010.

20.7.3 Table Specifications Do Not Exist (Type 3 JDBC only)

Exception: If you are using the Type 3 JDBC driver, you might receive an error message that indicates that table specifications do not exist.

Cause: This exception indicates that table specifications have not been generated for a particular table.

Recovery: To generate specifications for a table, sign-on to an HTML web client and run data browser for the table. When you use a Type 3 JDBC driver, you must run dataBrowser for any table that has not been previously opened from an HTM web client.

20.8 JDBC Driver Terminology

The following table provider terminology used in this chapter. These terms are not available in the glossary.

connection mode
Connection mode is a term that applies only to the JDBC drivers and provides an indication of the type of additional filtering and processing that the JD Edwards EnterpriseOne data that you are accessing requires. Application code designates a connection mode when it establishes each new connection.

connection properties
Properties that applications pass to the JDBC drivers when establishing a new connection in order to configure a particular connection type. The concept of connection properties is a standard JDBC mechanism, but each driver defines its own set of recognized connection properties.

connection URL
A string that identifies a particular data source to which to connect. The concept of a connection URL is a standard JDBC mechanism, but each driver defines its own URL syntax.

driver class name
A string that identifies the primary class for a JDBC driver. You must register this class name with the JDBC driver manager before using it. this is a standard JDBC concept, but each driver defines its own driver class name.

driver manager
The JDBC class that manages multiple registered JDBC drivers and dispatches connection initialization requests to them. the Java driver manager class is java.sql.DriverManager.

ERP data
Data that is managed within an ERP environment.

JDBj
The Java class library that encapsulates most aspects of JD Edwards EnterpriseOne software data access middleware functionality such as OCM, ERP triggers, ERP business views, ERP row security, and decimal scrubbing.

JDBC Type 3 driver
The JDBC Type 3 driver is a network-protocol, all-Java driver. This style of driver translates JDBC calls into the middleware vendor's protocol, which is then translated to a DBMS protocol by a middleware server. The middleware provides connectivity to many different databases.

JDBC Type 4 driver

The JDBC Type 4 driver is a pure Java driver that accesses a database directly. Type 4 drivers are sometimes called *thin* drivers. Type 4 JDBC drivers have relatively fast performance.

21

Using Oracle Orchestration Systems

This chapter contains the following topics:

- Section 21.1, "Understanding Oracle Orchestration Systems"
- Section 21.2, "Configuring Orchestration Cross References"
- Section 21.3, "Using Password Indirection (Optional)"
- Section 21.4, "Setting Up the Cross-Reference Java Binding Service"

21.1 Understanding Oracle Orchestration Systems

JD Edwards EnterpriseOne can provide and consume web services using Oracle's BPEL-PM and ESB orchestration systems. JD Edwards EnterpriseOne can also send event notifications to third-party systems using the BPEL-PM and ESB orchestration systems.

See *Oracle BPEL Process Manager Developer's Guide*.

See *Oracle ESB Developer's Guide*.

See *JD Edwards EnterpriseOne Tools Transaction Server Reference Guide* on My Oracle Support.

See "JD Edwards EnterpriseOne Tools Business Services Development Overview" in the *JD Edwards EnterpriseOne Tools Business Services Development Guide*.

21.1.1 BPEL-PM

You can use Oracle's BPEL-PM to create a web service. BPEL-PM provides high-performance, reliable execution of service-oriented business processes that are defined with the BPEL standards. BPEL-PM is primarily used to create integrated business processes. It contains native support of BPEL, SML, XSLT, XPATH, JMS, JCA, and web services.

BPEL is an XML-based language for enabling task-sharing across multiple enterprises using a combination of web services. BPEL is based on XML schema, simple object access protocol (SOAP), and web services description language (WSDL). BPEL provides enterprises with an industry standard for business process orchestration and execution. Using BPEL, you design a business process that integrates a series of discrete services into an end-to-end process flow.

You can consume JD Edwards EnterpriseOne web services in a BPEL business process using BPEL-PM. You can also use the JD Edwards EnterpriseOne transaction server to publish and deliver event notifications to BPEL-PM. In addition, you can consume a BPEL-PM web service in JD Edwards EnterpriseOne using the business services server. You use the JDeveloper BPEL Designer, a tool that is part of the Oracle BPEL-PM product, to build, deploy, and test a BPEL PM process that consumes JD Edwards EnterpriseOne web services.

21.1.2 ESB

Oracle Enterprise Service Bus (ESB) is a component of the Oracle Service-Oriented Architecture (SOA) suite delivering loosely coupled data and enterprise application integration. Oracle ESB features a multiprotocol message bus with centralized monitoring and management of distributed services where all services are exposed as standard web services. Oracle ESB contains message flows that use adapters, transformations, and routing rules to distribute data throughout and beyond the enterprise.

You can consume JD Edwards EnterpriseOne web services in an ESB flow. You can also use the JD Edwards EnterpriseOne transaction server to publish and deliver event notifications to ESB. In addition, you can consume an ESB web service in JD Edwards EnterpriseOne using the business services server. You use the JDeveloper ESB Designer to build, deploy, and test a BPEL PM process that consumes JD Edwards EnterpriseOne web services.

21.1.3 Orchestration

Orchestration enables you to map and transform data between two disparate systems. This allows data in a JD Edwards EnterpriseOne format to be transformed to the data format of another application or system and the data format of another application or system to be transformed to the data format of JD Edwards EnterpriseOne. Orchestration is a key component that enables you to plug and play different software modules to complete your integration solution. Much of the orchestration system is based on SOA.

Oracle provides applications that help you create cross-reference utilities. The cross-reference utilities are used within the orchestration system.

Note: If the orchestration system that you use provides features to do dynamic cross-referencing, static cross-referencing, or both, you should use them.

21.2 Configuring Orchestration Cross References

This section provides an overview of the orchestration cross-reference configuration and discusses how to:

- Register cross-reference XPATH functions in JDeveloper.
- Configure access to orchestration cross-reference APIs.
- Create a data source in OC4J.

21.2.1 Understanding the Orchestration Cross-Reference Configuration

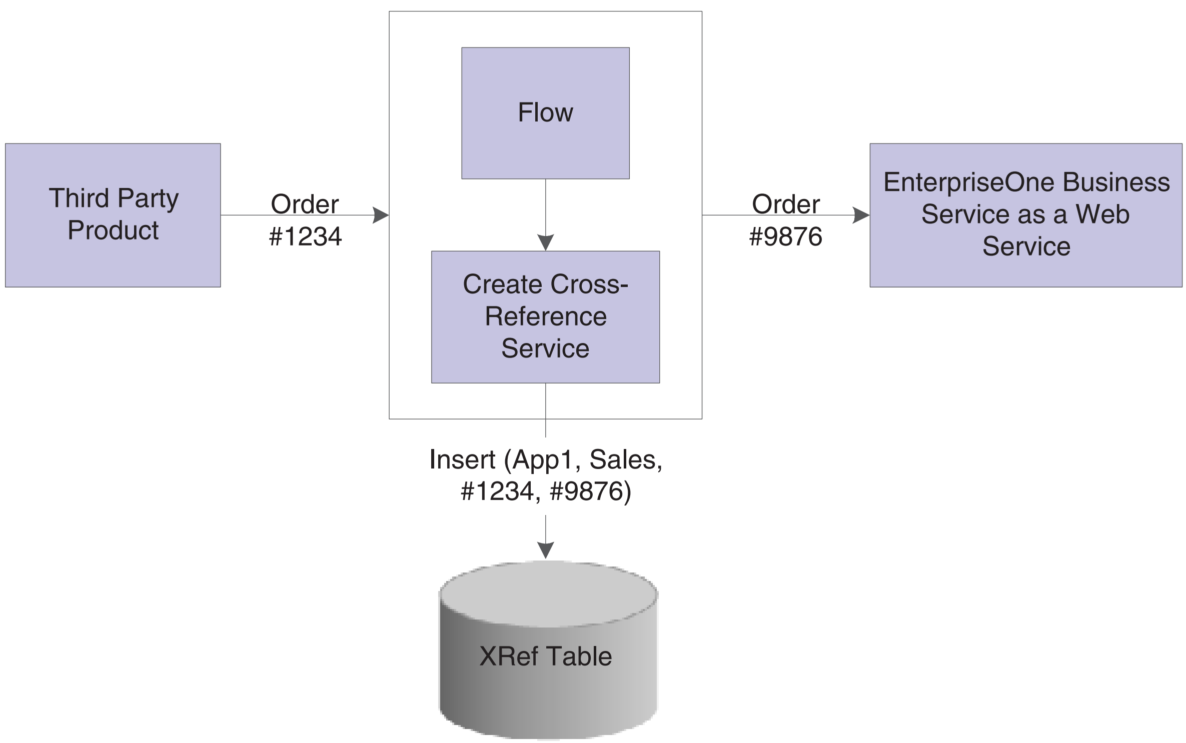
Orchestration cross-references are key/value data pairs that are used for referencing corresponding values between disparate systems. The orchestration system uses two types of cross-references: code and key. Code cross-references pertain to static references; for example, in JD Edwards EnterpriseOne, the country code US might be equivalent to USA in another software system. Key cross-references are dynamic cross-references that are added during orchestration runtime, such as sales orders across systems. For example, the sales order number 9876 in JD Edwards EnterpriseOne is equivalent to sales order number 1234 in a third-party application.

Cross-reference utilities provide a way to read, add, modify, and delete cross-reference entries at runtime. Before you can use the Orchestration Cross Reference Admin tool to add cross-references, you must perform preliminary tasks to configure the orchestration system. This chapter identifies the tasks that a system administrator and developer must perform to configure the system for cross-reference orchestration. The system administrator performs certain tasks (for example, cross-reference API access, OC4J data source creation, cross-reference Java binding service setup) to enable developers to add, modify, and delete cross-references. The developer registers cross-reference XPATH functions in JDeveloper. These configuration tasks enable interaction between JD Edwards EnterpriseOne and third-party applications in the orchestration system.

21.2.1.1 How the Orchestration System Uses Cross-References

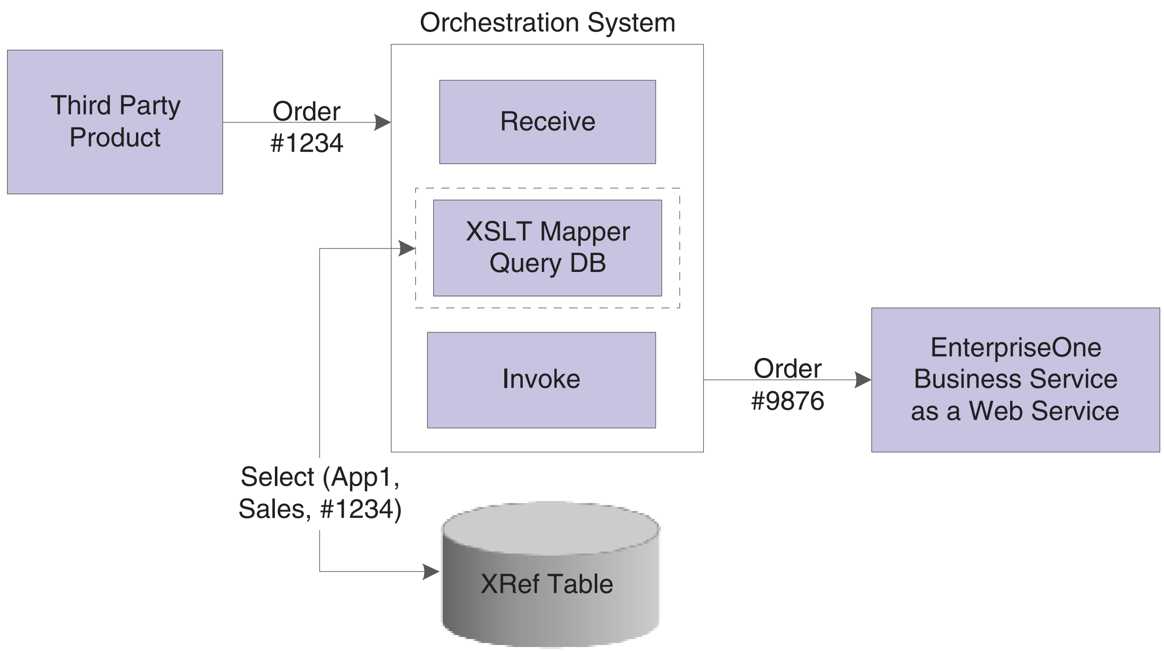
This diagram shows the dynamic cross-referencing in the orchestration process. In this scenario, a third-party application calls an orchestration system as part of order processing and passes the order number in the payload. The orchestration system calls a JD Edwards EnterpriseOne web service to create an order and to get an EnterpriseOne order number. Upon successful response from JD Edwards EnterpriseOne, the orchestration system creates and stores mapping information for the order numbers from the third-party system and JD Edwards EnterpriseOne.

Figure 21–1 Creating a cross-reference key



This diagram shows the orchestration system dynamically updating an existing cross-reference. In this scenario, a third-party application calls the orchestration system as part of updating or deleting an existing order and sends the order number in the payload. The orchestration system gets the JD Edwards EnterpriseOne order number from the cross-reference table and calls a JD Edwards EnterpriseOne web service to update or delete the order.

Figure 21–2 Getting a cross-reference key



21.2.1.2 Common Notations and Variables in This Document

You should be familiar with these notations and variables before performing the tasks in this chapter:

Notation / Variable	Definition
ORACLE_HOME	Orchestration tool install directory.
OC4J_HOME	Orchestration tool OC4J directory. Typical values include: ▪ BPEL – ORACLE_HOME/bpel/system/appserver/ocrj/j2ee/home ▪ ESB – ORACLE_HOME/j2ee/home
JDEVELOPER_HOME	JDeveloper home directory. This must be JDeveloper with the BPEL/ESB extensions MTR.

21.2.2 Registering Cross-Reference Read XPATH Functions in JDeveloper

Cross-reference Read XPATH functions provide common syntax and semantics for addressing parts of an XML document. To access Cross-Reference Read XPATH functions, you must register the XPATH functions in JDeveloper. The XPATH functions reside in the CrossReferenceRead.xml file. You must register the XPATH functions for both JDeveloper and runtime.

Note: This is a one-time setup that a developer performs.

To register the XPATH functions:

1. Copy the CrossRefenceRead.xml file and paste it in the JDEVELOPER_HOME directory:
2. Open JDeveloper.

   //delivery/production/Orchestration/WSDL_Schema

   In Microsoft Windows, this file is located in:

   //system/classes/Orchestration/WSDL_Schema
3. On the Preferences form, navigate to Tools, Preferences, XSL Maps.
4. In the User Defined Extension Functions Config File field, enter the path to the CrossReferenceRead.xml file, and then click OK.
5. Restart JDeveloper.

21.2.3 Configuring Access to Orchestration Cross-Reference APIs

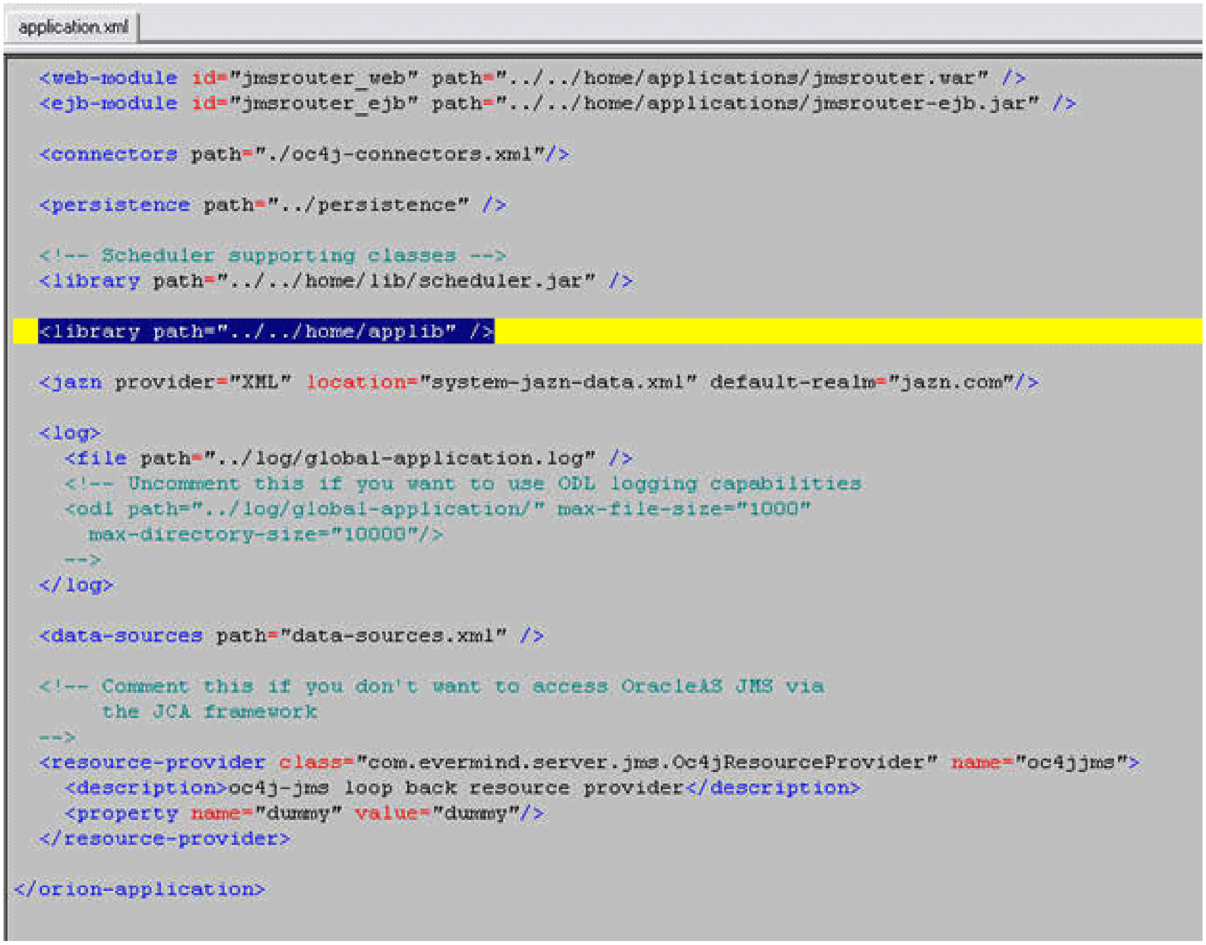
For the orchestration system to access the cross-reference APIs, you must register the XPATH function in JDeveloper for runtime.

Note: The orchestration cross-reference APIs are for use with orchestration cross-references only; they are not used for business service properties.

This is a one-time setup that an administrator performs.

This example XML document is referred to in Step 2 of the task:

Figure 21–3 XML document for registering XPATH functions in JDeveloper



application.xml

```
  <web-module id="jmsrouter_web" path="../../home/applications/jmsrouter.war" />
  <ejb-module id="jmsrouter_ejb" path="../../home/applications/jmsrouter-ejb.jar" />

  <connectors path="./oc4j-connectors.xml"/>

  <persistence path="../persistence" />

  <!-- Scheduler supporting classes -->
  <library path="../../home/lib/scheduler.jar" />

  <library path="../../home/applib" />

  <jazn provider="XML" location="system-jazn-data.xml" default-realm="jazn.com"/>

  <log>
    <file path="../log/global-application.log" />
    <!-- Uncomment this if you want to use ODL logging capabilities
    <odl path="../log/global-application/" max-file-size="1000"
      max-directory-size="10000"/>
    -->
  </log>

  <data-sources path="data-sources.xml" />

  <!-- Comment this if you don't want to access OracleAS JMS via
       the JCA framework
  -->
  <resource-provider class="com.evermind.server.jms.Oc4jResourceProvider" name="oc4jjms">
    <description>oc4j-jms loop back resource provider</description>
    <property name="dummy" value="dummy"/>
  </resource-provider>

</orion-application>
```

To register the XPATH functions in JDeveloper for runtime.

1. Copy FS_SCCR_JAR from the following location and paste it in the OC4J_HOME/applib directory:

   //delivery/production/orchestration

   In Microsoft Windows, this file is located in:

   //system/classes/orchestration

2. Open OC4J_HOME/config/application.xml in a text editor.

3. If the following line is not already in the XML document, add it between the start and end tags of the orion-application:

<library path="../../home/applib" />

4. Restart OC4J.

21.2.4 Creating a Data Source in OC4J

Cross-reference access APIs connect to the JD Edwards EnterpriseOne database using the JNDI data source defined in OC4J. A system administrator must define the data source based on the database platform being used. Before you can define the data source, you must create the data source in OC4J.

To create a data source in OC4J:

1. Open OC4J_HOME/config/application.xml in a text editor.
2. Add the following line between the start and end tags of the orion-application if it is not already there:

   <library path="../../home/applib" />
3. Place the required database driver in OC4J_HOME/applib directory.
4. Open OC4J_HOME/config/data-sources.xml in text editor.
5. Add native data source definition for the database platforms as described in this section.
6. Restart OC4J.

After you create the data source in OC4J, you can define it for any one of these database platforms:

- Oracle
- UDB/DB2 for IBM i
- DB2 for IBM i
- SQL Server

21.2.4.1 Defining an Oracle Data Source in OC4J

Use the Oracle JDBC driver provided with OC4J. If you prefer a different driver, place the driver in the applib directory.

```
<native-data-source user="user" password="password"
   url="jdbc:oracle:thin:@//dbserver:dbport/SID"
   data-source-class="oracle.jdbc.pool.OracleDataSource"
   jndi-name="jdbc/abcoraDS" name="abcoraDS"/>
```

21.2.4.2 Defining a UDB/DB2 for IBM i Data Source in OC4J

Use UDB universal driver db2java.zip. This is a type 2 jdbc driver. The UDB client must be present and the database must be cataloged for the connection to work.

```
<native-data-source user="user" password="password"
   url="jdbc:db2:dbname"
   data-source-class="COM.ibm.db2.jdbc.DB2DataSource"
   jndi-name="jdbc/udbDS" name="udbDS">
       <property name="databaseName" value="dbname"/>
       <property name="serverName" value="dbservername"/>
</native-data-source>
```

21.2.4.3 Defining a DB2 for IBM i Data Source in OC4J

Use IBM i access driver version 7.0 or greater

The jt400.jar must be in the applib directory.

```
<native-data-source user="user" password="password"
   url="jdbc:as400://as400machine"
   data-source-class="com.ibm.as400.access.AS400JDBCDataSource"
   jndi-name="jdbc/asDS" name="asDS">
      <property name="databaseName" value="dbname"/>
      <property name="serverName" value="servername"/>
      <property name="libraries" value="database library to access"/>
</native-data-source>
```

21.2.4.4 Defining a SQL Server Data Source in OC4J

This data source definition is for use with the Microsoft JDBC driver. These files must be in the applib directory:

- msbase.jar
- msutil.jar
- mssqlserver.jar

```
<native-data-source user="username" password="password"
   url="jdbc:microsoft:sqlserver://databaseserver:TCP/IP port;databaseName=DBName"
   data-source-class="com.microsoft.jdbcx.sqlserver.SQLServerDataSource"
   jndi-name="jdbc/abcDS" name="abcDS">
        <property name="databaseName" value="DBName"/>
        <property name="serverName" value="servername\databaseinstancename"/>
</native-data-source>
```

21.3 Using Password Indirection (Optional)

This section provides an overview of password indirection and provides the tasks for setting up password indirection in OC4J, which include how to:

- Edit the default application.xml to use system-jazn data.
- Add a JAZN user.
- Add password indirection in the data source.

Note: Oracle recommends setting up password indirection for preventing security vulnerabilities. However, it is not required.

21.3.1 Understanding Password Indirection

The data-sources.xml file requires passwords for authentication. Embedding these passwords without some kind of obfuscation poses a security risk. To avoid this problem, OC4J supports password indirection.

An indirect password is made up of a special indirection symbol (->) and a user name (or user name and realm). When OC4J encounters an indirect password, it retrieves the password associated with the specified user from the security store provided by a user manager.

21.3.2 Editing the Default Application.xml to Use System-jazn Data

You must edit the default application.xml file to use system-jazn data.

To edit the default application.xml to use system-jazn data:

1. Open OC4J_HOME/config/application.xml in a text editor.
2. Find this line, <jazn provider="XML"/>, and replace it with this line:

   <jazn provider= ="XML" location="system-jazn-data.xml"
   default-realm="jazn.com"/>

21.3.3 Adding a JAZN User

To add a JAZN user:

1. Enter the following URL in a web browser to open OC4J EM:

   http://oc4jserver:port/em
2. Navigate to Administration, Security, Security providers.
3. On the Security Providers page, click the Instance Level Security button and then select the Realms tab.
4. On the Instance Level Security page, click the link in the Users column for the default realm jazn.com.
5. On the Users page, click the Create button and then complete these fields on the Add User page:
   - Username
   - Password
   - Confirm Password

21.3.4 Adding Password Indirection in the Data Source

To add password indirection in the data source:

1. Open the data-sources.xml file in a text editor.
2. Find the data source definition to edit.
3. In the password field for the data-source, enter the indirection symbol (->) followed by the JAZN user that you added, for example:

   ->jazn.com/xrefuser

   The data source definition should look like this when complete:

```
<native-data-source user="xrefuser" password="->jazn.com/xrefuser"
     url="jdbc:microsoft:sqlserver://localhost:1050;databaseName=JDELocal"
     data-source-class="com.microsoft.jdbcx.sqlserver.SQLServerDataSource"
     jndi-name="jdbc/eoneDS" name="eoneDS">
        <property name="databaseName" value="JDELocal"/>
        <property name="serverName" value="localhost\JDELocal"/>
  </native-data-source>
```

4. Save the file and restart OC4J.

21.4 Setting Up the Cross-Reference Java Binding Service

The cross-reference Java binding service provides access to cross-reference utilities from the orchestration system. You can register the Java binding service using either of these methods:

- Registering in the shared library (BPEL-PM or ESB)

- Putting Java binding classes in BPEL-PM classpath (BPEL-PM only)

This service is delivered as two files: CrossReferenceJavaBinding.wsdl and CrossReferenceAccess.xsd. You can locate the files in this directory:

//delivery/production/Orchestration/WSDL_Schema

In Microsoft Windows client, the files are located in this directory:

/system/classes/Orchestration/WSDL_Schema

21.4.1 Registering the Java Binding Service

To register the Java binding service in the BPEL-PM or ESB shared library:

1. Copy the FIS_SCCR_JAR.jar to the following directory:
   - For BPEL-PM:

     ORACLE_HOME/bpel/system/classes
   - For ESB:

     ORACLE_HOME/integration/esb/lib
2. Copy the CrossReferenceJavaBinding.wsdl and CrossReferenceAccess.xsd files to the ORACLE_HOME directory.
3. Open the server.xml file in a text editor. You can locate this file in OC4J_HOME/config folder of BPEL/ESB.
4. In the server.xml file, locate shared-library name="oracle.bpel.common".
5. Add FIS_SCCR_JAR.jar in this shared library to the following location:
   - For BPEL-PM:

     <code-source path="D:\OraBPEL_1\bpel/system/classes/FIS_SCCR_JAR.jar"/>
   - For ESB:

     <code-source path="../../../integration/esb/lib/FIS_SCCR_JAR.jar"/>

21.4.2 Placing Java Binding Classes in the Classpath

To place the Java binding classes in the classpath for either BPEL-PM or ESB.

1. Unzip the contents of the FIS_SCCR_JAR.jar file to this directory:

   ORACLE_HOME/bpel/system/classes
2. Copy the provided CrossReferenceJavaBinding.wsdl and CrossReferenceAccess.xsd to the ORACLE_HOME directory.

21.4.3 Using Cross-Reference Read Services from XSL Mapper

The JD Edwards EnterpriseOne read services for cross-references are available from XSL Mapper in both BPEL and ESB. The steps in this section describe how to use these read services.

To use the cross reference read service from XSL Mapper:

1. Invoke XSL Mapper.

   In BPEL-PM, add the Transform activity to the BPEL-PM flow and provide the input and output document.

In ESB, use the transform function in routing rules for any ESB service.

2. In the XSL Mapper, from the Component Palette drop-down menu, select the User Defined Extension Functions component.

   The system should display two functions: getCrossReferenceE1Val and getCrossReferenceThirdPartyVal.

3. Select the required function and drag it to the mapping area.
4. Define inputs for the function using any one or a combination of these methods:
   - Double-click the function and provide string inputs using double or single quotes.
   - Double-click the function and provide the XPATH expression as input.
   - Link elements from input schema to function inputs.
5. Map the output to the required element in output schema.
6. Save the XSL Map.

21.4.4 Using JD Edwards EnterpriseOne Cross-Reference Services

You can use the JD Edwards EnterpriseOne Cross Reference Service to add, modify, and delete cross-reference information.

21.4.4.1 BPEL-PM

You can use the JD Edwards EnterpriseOne Cross Reference Service to add, modify, and delete cross-reference information in Oracle's BPEL-PM orchestration system. The BPEL project must be created in JDeveloper BPEL-PM designer.

To create a partner link in JDeveloper BPEL-PM designer using the service WSDL file.

1. In the JDeveloper BPEL-PM designer, select the PartnerLink activity and drag it to the BPEL-PM process page.
2. On the Create Partner Link dialog box, enter the name of the WSDL file for the Java binding service:

   ORACLE_HOME/CrossReferenceJavaBinding.wsdl

3. Click Yes when asked to make a local copy of the file.
4. Click Yes when asked to create a new Partner Link type.
5. From the Partner Role drop-down menu, select the default generated role.
6. Click OK.

The system creates the Partner Link. The Partner Link can now be used to invoke cross-reference services.

21.4.4.2 ESB

You can use the JD Edwards EnterpriseOne Cross Reference Service to add, modify, and delete cross-reference information in ESB. You use ESB JDeveloper.

To invoke cross-reference services from ESB:

1. Import web service schema EoneXrefAccessWSSchema.xsd in the project using the File, Import utility.

   The xsd file should have been copied to your ORACLE_HOME directory during the web service deployment step.

2. Right-click the ESB system design page and select Create ESB Service, SOAP Service.
3. Name the service **EOneXrefAccessWS.**
4. Select ESB System/group if you are not using the default.
5. For the Java binding service, enter the WSDL file as provided in the example:

   ORACLE_HOME/CrossReferenceJavaBinding.wsdl
6. Click OK.

   The system creates the SOAP service and displays all available operations.
7. To create the routing service for the required operation, right-click the system page and select Create ESB Service, Routing Service.
8. Enter an appropriate name for the service, such as EOneXrefAddRec_RS.
9. Select ESB System/group if you are not using the default.
10. Click the Generate WSDL from the Schemas option.
11. On the Request tab, for Schema Location, click Browse.
12. Select the operation input element from EoneXrefAccessWSSchema.xsd under Project Schema Files and click OK.
13. On the Reply tab, repeat the preceding steps and select the response element for the chosen operation, and then click OK.
14. Double-click Routing service, expand the Routing Rules tab, and then click the Add button.
15. Select the web service operation as the target service.
16. Add filter rules and transformations as needed.
17. Save the project.

This routing service can be called from other ESB services for executing cross-reference operations.

22 Setting Up Orchestration Cross-References

This chapter contains the following topics:

22.1 Understanding Orchestration Cross-References

Orchestration cross-references (hereafter referred to as *cross-references*) are key/value data pairs used in the orchestration system. You add a cross-reference to associate a JD Edwards EnterpriseOne value, such as an Address Book number, with the equivalent value in a third-party application. For example, a third-party application that is integrated with JD Edwards EnterpriseOne might contain a field called Client Number that equates to the Customer Number field in JD Edwards EnterpriseOne. To share this data between the two systems, you create a cross-reference record that associates Client Number with Customer Number.

The Business Service Cross Reference program (P952000) is the JD Edwards EnterpriseOne program that enables you to manage cross-references.

In JD Edwards EnterpriseOne, you can define a cross-references in one of these categories:

- Code

  A code reference pertains to static items in JD Edwards EnterpriseOne, such as a field or user-defined code. For example, Address Book Number is a code reference. You can use P952000 to add, customize, or delete code references.

- Key

  A key reference contains transactional information that is added during orchestration runtime. For example, a key code might map the sales order number 9876 in JD Edwards EnterpriseOne to the equivalent sales order number in a third-party application. You can use P952000 to add, modify, or delete key references.

22.1.1 Code and Key Cross-Reference Categorization

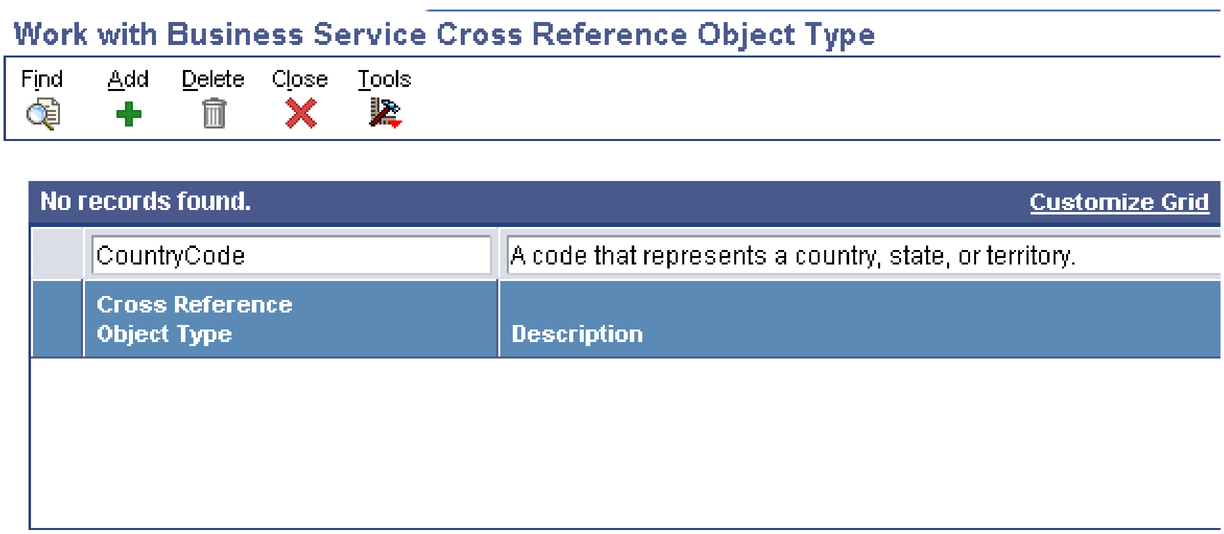
JD Edwards EnterpriseOne uses cross-reference object types to categorize cross-references. You use cross-reference object types to group together code and key cross-references of similar type. For example, you can add cross-reference object types called *countrycode,unitofmeasure,* and *purchaseordernumber.* You associate each cross-reference that you add to the appropriate cross-reference object type, which serves as a category for a particular group of cross-references.

Before you add cross-references to the system, you should analyze the fields and data that you are cross-referencing and define a categorization system that you can use to group cross-references into categories. This categorization helps you manage cross-references so that you can readily review, modify, and remove cross-references as needed. You can set up all the cross-reference object types in JD Edwards EnterpriseOne before you add cross-references to the system, or add additional cross-reference object types as needed.

22.2 Adding Cross-Reference Object Types

JD Edwards EnterpriseOne requires that you assign each cross-reference to a cross-reference object type. Cross reference object types enable you to group cross-references by category. Therefore, you must add cross-reference object types before you add cross-references.

Figure 22–1 Work with Business Service Cross Reference Object Type form



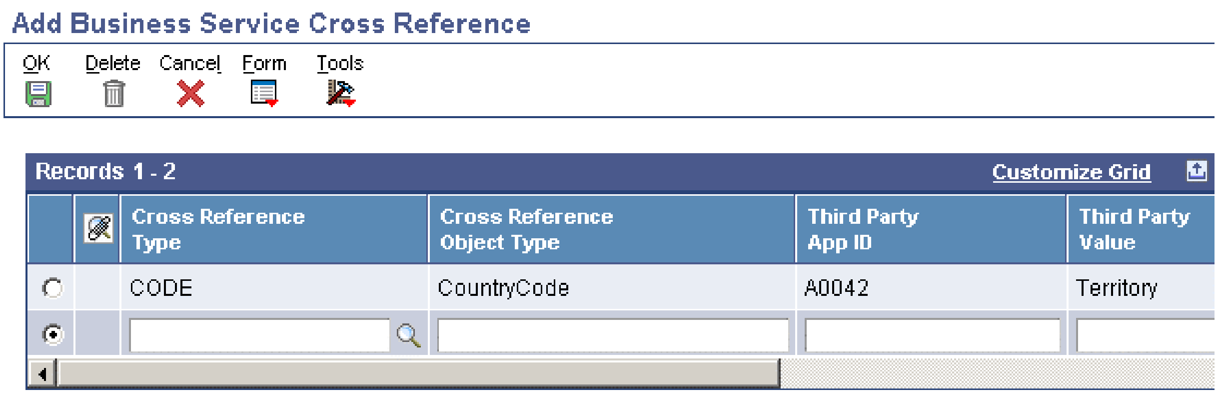
To add a cross-reference object type:

1. To access the Work with Orchestration Cross Reference form, enter **P952000** in the Fast Path field.
2. From the Form menu, select Object Type.
3. On the Work with Orchestration Cross Reference Object Type form, click the Add button.
4. On the Add Orchestration Cross Reference Object Type form, in the Cross Reference Object Type field, enter a name that you want to use to categorize cross-references.
5. In the Description field, enter a description that defines the purpose of the cross-reference object type, and then click the OK button.

22.3 Adding Orchestration Cross-References

You add orchestration cross-references to assign JD Edwards EnterpriseOne values to values in a third-party application.

Figure 22–2 Add Business Service Cross Reference form



1. To access the Work with Orchestration Cross Reference form, enter **P952000** in the Fast Path field.
2. Click the Add button.
   a. On the Add Orchestration Cross Reference form, add a cross-reference record by entering a value for each of these columns in the grid:
      – **Orchestration Cross Reference Type**

        Click the search button to select either CODE or KEY as the orchestration cross-reference type.

      – **Object Type**

        Click the search button to select a cross-reference object type that you want to use to categorize the cross-reference. If no suitable object type is available, you can add one in P952000.

      – **Third Party App ID**

        Enter an external system identifier, also known as a third-party application ID, to identify the system outside JD Edwards EnterpriseOne to which the cross-reference external value belongs, for example PeopleSoft CRM, E-Business Suite.

      – **Third Party Value**

        Enter a value from the external system that requires cross-referencing to an equivalent value in JD Edwards EnterpriseOne.

      – **EOne Value**

        Enter a JD Edwards EnterpriseOne value that is cross-referenced to the value in the external system.

   b. Press the Tab key to add additional cross-references as needed, and then click the OK button when complete.

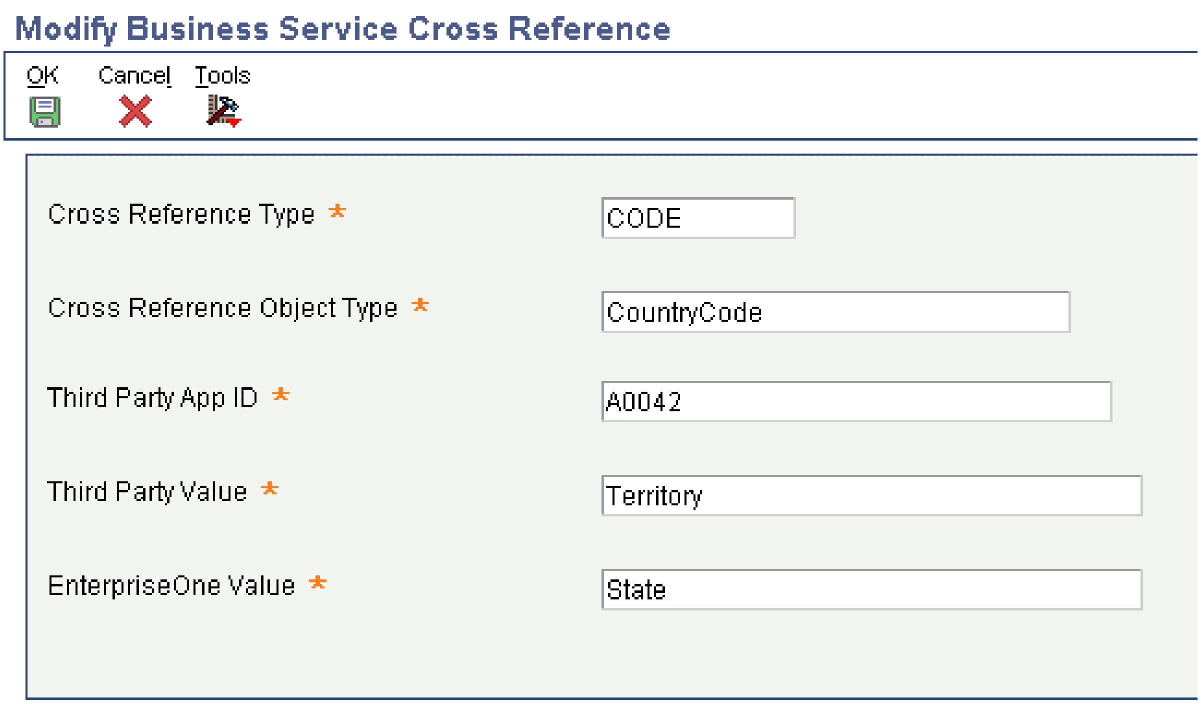
      When you click the OK button, the system saves the cross-reference records to the appropriate tables. You can review the records in the Work with Orchestration Cross Reference form.

22.4 Reviewing or Modifying Orchestration Cross-References

In P952000, you can search for and review all of the current cross-reference records in the system. You can also view a particular subset of cross-reference records by searching on either key or code cross-references. You can further refine the search so that the system displays only records that belong to a particular cross-reference object type.

In addition to reviewing current cross-reference records, P952000 enables you to modify cross-references. You can modify any of the values that make up the cross-reference, including changing the reference type from code to key or vice versa.

Figure 22–3 Modify Business Service Cross Reference form



To view or modify cross-references:

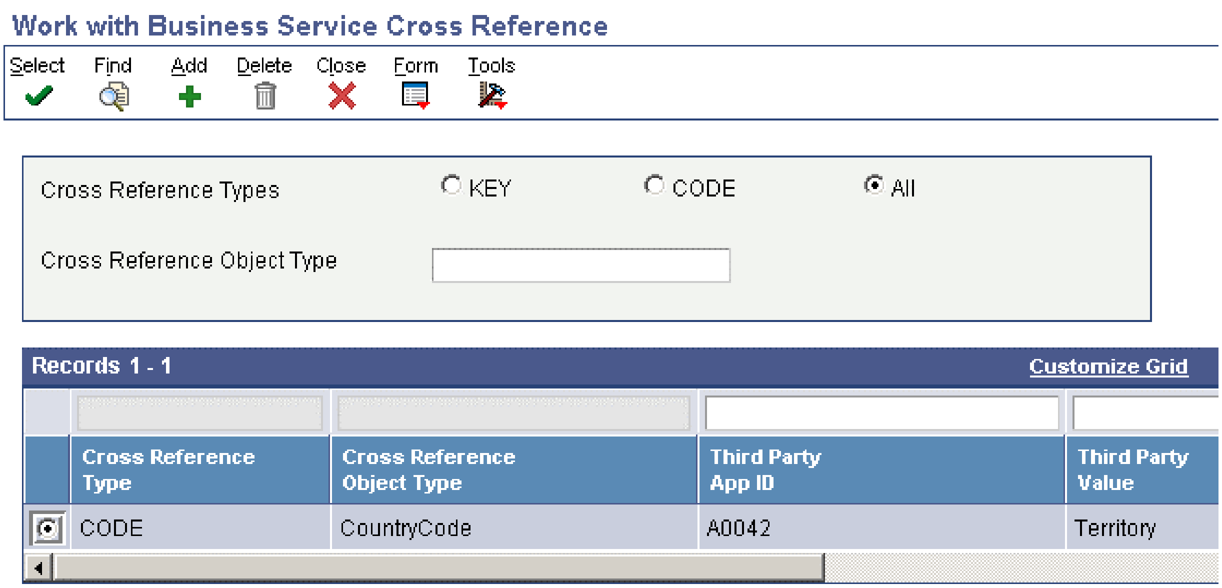
1. To access the Work with Orchestration Cross Reference form, enter **P952000** in the Fast Path field.
2. Click the appropriate Orchestration Cross Reference Types option to view all cross-reference records, key cross-references, or code cross-references, and then click the Find button.
3. To further refine the search, enter a cross-reference object type in the Cross Reference Object Type field, and then click the Find button.
4. To modify a cross-reference, highlight the row that contains the cross-reference and then click the Select button.
5. On the Modify Orchestration Cross Reference form, modify any of these fields as appropriate, and then click the OK button:

– Reference Type
– Object Type
– Third Party App ID
– Third Party Value
– EnterpriseOne Value

22.5 Deleting Orchestration Cross-References

If a cross-reference becomes obsolete and is no longer necessary, you can delete it.

Figure 22–4 Work with Business Service Cross Reference form



To delete a cross-reference:

1. To access the Work with Orchestration Cross Reference form, enter **P952000** in the Fast Path field.
2. Search for the cross-reference record that you want to delete.
3. Highlight the row for the record, and click the Delete button

A

Classic Events

This appendix contains the following topics:

Note: This chapter is applicable only if you use classic events delivery. Classic event delivery is available when you use JD Edwards EnterpriseOne Tools 8.93 or earlier releases, or if you use JD Edwards EnterpriseOne Tools 8.94 with JD Edwards EnterpriseOne Applications 8.10.

Refer to the Guaranteed Events chapters if you use JD Edwards EnterpriseOne Tools 8.94 with JD Edwards EnterpriseOne Applications 8.11 or JD Edwards EnterpriseOne Tools 8.95 and later tools releases with JD Edwards EnterpriseOne Applications 8.10 and later Applications releases.

A.1 Understanding Classic Events

JD Edwards event functionality provides an infrastructure that can capture JD Edwards EnterpriseOne transactions in various ways and provide real-time notification to third-party software, end users, and other JD Edwards systems, such as WSG and CRM.

JD Edwards EnterpriseOne notifications are called events. The JD Edwards EnterpriseOne event system implements a publish/subscribe model. Events are delivered to subscribers in XML documents that contain detailed information about the event. For example, when you enter a sales order into the system, the system can automatically send the sales order information to a CRM or supply chain management application for further processing. If your system is IBM, you can use IBM WebSphere MQ messaging to receive events. If your system is Microsoft, you can use MSMQ messaging to receive events. IBM WebSphere MQ and MSMQ provide a point-to-point interface with JD Edwards EnterpriseOne. JD Edwards EnterpriseOne supports three kinds of events, as described in the table:

Type of Event	Description
Z Events	A service that uses interface table functionality to capture JD Edwards EnterpriseOne transactions and provide notification to third-party software, end-users, and other JD Edwards systems that have requested to be notified when certain transactions occur.
Real-Time Events	A service that uses system calls to capture JD Edwards EnterpriseOne transactions as they occur and provide notification to third-party software, end users, and other JD Edwards systems that have requested notification when certain transactions occur.
XAPI Events	A service that uses system calls to capture JD Edwards EnterpriseOne transactions as they occur and then calls third-party software, end users, and other JD Edwards systems that have requested notification when the specified transactions occur to return a response. XAPI events can be from JD Edwards EnterpriseOne to a third-party system, from a third-party system to JD Edwards EnterpriseOne, or between two JD Edwards EnterpriseOne systems.

The JD Edwards EnterpriseOne event system consists of these modules:

- Event distributor
- Event generators
- Transport drivers

The event distributor is a JD Edwards EnterpriseOne kernel process called the event notification (EVN) kernel. The EVN kernel manages the subscribers and notifies them when an event occurs. The EVN kernel is shared by Z events, real-time events, and XAPI events.

Event generators are processes or libraries capable of generating events. JD Edwards EnterpriseOne provides three default event generators:

- Z event generator, which generates Z events.
- Real-time event generator, which generates real-time events.
- XAPI event generator, which generates XAPI events.

Z events, real-time events, and XAPI events have slightly different XML documents.

The event distributor uses a transport driver to send events. JD Edwards EnterpriseOne provides a default transport driver that uses JDENET. The event distributor can also send event documents to designated IBM WebSphere MQ or MSMQ outbound queues using IBM WebSphere MQ or MSMQ transport drivers. If you use IBM WebSphere MQ or MSMQ transport drivers to receive events, you receive all events that are defined in the F90701 table.

A.2 Defining Events

When an event is generated, the IEO kernel reads the F90701 table for that event. If the specified event category is different from the event category configured in the database, the system writes an error to the IEO log file. If the database definition of the event is not found, the system writes this message to the IEO log.

```
Warning: table F90701 doesn't exist. Some features will be turned off.
```

> **Note:** When you update or modify the F90701 table, you must restart the server for the changes to become effective.

A.2.1 Reducing Network Traffic

To reduce network traffic, real-time event processing sends only active events. If a single event is identified as inactive in table F90701 and is part of an active container event, the CallObject kernel sends the active container event and the active single events to the IEO and EVN kernels to create the XML and to distribute to subscribers. Inactive single events that have been disabled by the CallObject kernel are embedded in the container event but are not sent as separate single events.

The CallObject kernel debug log contains information about the inactive single events that are not created. This is an example CallObject Kernel debug log message:

```
Inactive container event <event name> is not added to prevent bursting
```

This scenario illustrates the process:

- RTABHDR and RTABPHOUT are inactive single events.
- RTABEAOUT is an active single event.
- RTABOUT is an active container event that contains RTABHDR, RTABPHOUT, and RTABEAOUT.
- The business function creates these events:
  - 1 RTABHDR event
  - 2 RTABPHOUT events
  - 5 RTABEAOUT events
- The RTABOUT container event and the five RTABEAOUT events are sent from the CallObject kernel to the IEO and EVN kernels for processing and distribution to the subscriber. Inactive single events, RTABHDR and RTABPHOUT, are not sent.
- The subscriber receives:
  - 1 RTABOUT container event with all of the subdata structures that are defined in the single events RTABHDR, RTABPHOUT, and RTABEAOUT.
  - 5 RTABEAOUT single events.

A.3 Subscribing to Events

For XAPI events, you must update the F90702 table so that you can receive a response to your XAPI event. Each XAPI event must have a logical subscriber, which you might have to set up. For Z and real-time events, the system dynamically updates this table when the event is created. You can use the F90702 table to view the persistent subscriptions for your Z and real-time events.

If the database table is missing, the system writes these messages to the IEO log:

```
CheckTableExists failed: invalid hEnv or hUser.
Warning: table F90702 doesn't exist. Some features will be turned off.
```

A.4 Configuring the jde.ini file for Events

The JD Edwards EnterpriseOne server jde.ini file must be properly configured to support Z, real-time, and XAPI event generation. You use a text editor to manually edit and verify specific settings in the JD Edwards EnterpriseOne server jde.ini file.

Note: If your enterprise contains more than one JD Edwards EnterpriseOne server, you must ensure that each server has the same settings for all logic, batch, and interoperability sections.

Use these kernel and [JDEITDRV] settings to configure the jde.ini file on your JD Edwards EnterpriseOne server. Configure the kernels that are appropriate for the type of event (Z, real-time, or XAPI) that you want to generate.

Note: To determine which kernels you need to set and for other jde.ini settings for each specific type of event, refer to the Configure the jde.ini File topics in the Events section of the Interoperability Guide.

A.4.1 [JDENET_KERNEL_DEF19]

Use these settings for a Windows Microsoft platform:

```
krnlName=EVN KERNEL
dispatchDLLName=jdeie.dll
dispatchDLLFunction=_JDEK_DispatchITMessage@28
maxNumberOfProcesses=2
numberOfAutoStartProcesses=2
```

A.4.2 [JDENET_KERNEL_DEF20]

Use these settings for a Microsoft Windows platform:

```
krnlName=IEO KERNEL
dispatchDLLName=jdeieo.dll
dispatchDLLFunction=_JDEK_DispatchIEOMessage@28
maxNumberOfProcesses=2
numberOfAutoStartProcesses=2
```

Important: If you use JD Edwards EnterpriseOne 8.10 or a release prior to JD Edwards EnterpriseOne 8.10, the maxNumberOfProcesses and the numberOfAutoStartProcesses settings for both the EVN kernel (JDENET_KERNEL_DEF19) and the IEO kernel (JDENET_KERNEL_DE20) should have the same value. For example, maxNumberOfProcesses=3 and numberOfAutoStartProcesses=3. This causes the processes to be automatically started and avoids the cyclic dependency of the three-way request message from the IEO to the EVN kernel with the Get Event List message from the EVN to the IEO kernel.

A.4.3 [JDENET_KERNEL_DEF22]

Use these settings for a Microsoft Windows platform:

```
krnlName=XML Dispatch KERNEL
```

```
dispatchDLLName=xmldispatch.dll
dispatchDLLFunction=_XMLDispatch@28
maxNumberOfProcesses=1
numberOfAutoStartProcesses=0
```

A.4.4 [JDENET_KERNEL_DEF24]

Use these settings for a Microsoft Windows platform:

```
krnlName=XML Service KERNEL
dispatchDLLName=xmlservice.dll
dispatchDLLFunction=_XMLServiceDispatch@28
maxNumberOfProcesses=1
numberOfAutoStartProcesses=0
```

A.4.5 [JDEITDRV]

Use these settings for a Microsoft Windows platform:

```
DrvCount=5
Drv1=Z:zdrv.dll
Drv2=RT:rtdrv.dll
Drv3=JDENET:jdetrdrv.dll
Drv4=MSMQ:msmqrtdrv.dll
Drv5=MQS:mqsrtdrv.dll
```

Note: You set event generation and transport drivers in the [JDEITDRV] section of the jde.ini file. You are not required to set all of these drives. For example, if you do not use messaging transports, you would not use the MSMQ and MQS settings. Be sure that you define DrvCount with the number of settings that you are using.

A.4.6 [JDENET]

This setting specifies the maximum number of JDENET kernels:

```
MaxKernelRanges=27
```

Note: You must set this value to encompass the total number of kernels that you defined.

This table shows the DLL and DRV settings for other platforms:

Table Column Heading	IBM i	HP9000	Sun or RS6000
EVN (19) dispatchDLLName	JDEIE	libjdeie.sl	libjdeie.so
EVN (19) dispatchDLLFunction	JDEK_ DispatchITMessage	JDEK_ DispatchITMessage	JDEK_ DispatchITMessage
IEO (20) dispatchDLLName	JDEIEO	libjdeieo.sl	libjdeieo.so
IEO (20) dispatchDLLFunction	JDEK_ DispatchIEOMessage	JDEK_ DispatchIEOMessage	JDEK_ DispatchIEOMessage

Table Column Heading	IBM i	HP9000	Sun or RS6000
XML Dispatch (22) dispatchDLLName	XMLDSPATCH	libxmldispatch.sl	libxmldispatch.so
XML Dispatch (22) dispatchDLLFunction	?XMLDispatch?	?XMLDispatch?	?XMLDispatch?
XML Service (24) dispatchDLLName	XMLSERVICE	libxmlservice.sl	libxmlservice.so
XML Service (24) dispatchDLLFunction	JDEK_ XMLServiceDispatch	JDEK_ XMLServiceDispatch	JDEK_ XMLServiceDispatch
Drv1	RTDRV	librtdrv.sl	librtdrv.so
Drv2	ZDRV	libzdrv.sl	libzdrv.so
Drv3	JDETRDRV	libjdetrdrv.sl	libjdstrdrv.so
Drv4	MSMQRTDRV	libmsmqrtdrv.sl	libmsmqrtdrv.so
Drv5	MQSRTDRV	libmqsrtdrv.sl	libmqsrtdrv.so

A.5 Using Reliable Event Delivery

This section provides an overview of reliable event delivery and discusses:

- Configuration of the reliable event delivery system.
- Use reliable event error messages.
- Minimize duplicate and lost events.
- Increase performance.
- Configuration of the jde.ini file.

A.5.1 Understanding Reliable Event Delivery

Reliable event delivery supports Z events, real-time events, and XAPI events. To use the Reliable Event Delivery feature, you must define your events in database tables. You cannot define your events in the jde.ini file.

The JDENET transport delivers Z events, real-time events, and XAPI events. Reliable event delivery ensures recovery and delivery of an event when transport problems arise, including some network problems. These scenarios identify circumstances for which events might be lost, but can be recovered and delivered:

- The JDENET process is down.
- JDENET fails to deliver because the network link between sender and receiver is permanently down.
- JDENET fails to deliver because the IPC buffer of the receiving kernel is full (sender and receiver are on different boxes).

Note: Reliable delivery covers failures that are related only to the transport of the events. Reliable delivery does not provide a once-and-only-once type of guarantee. Events might be lost and not recovered (or duplicates might be redelivered) in the presence of process failures (client and server).

Real-time event delivery is reliable for transportation failures between the real-time API and the Java connector, which includes IEO and EVN kernels. XAPI outbound event delivery is reliable for transportation failures between the XAPI API and the Java connector, including the IEO and EVN kernels. Z event delivery is reliable for transportation failures between the Z event generator and the Java connector.

The level of reliability is configurable based on whether the event is reliable or volatile. Volatile events are events that might be lost if the network or process fails and delivery is not reliable. Reliable events could be lost in the case of process failures only. You can configure the level of reliability for every event type. The level of reliability depends on whether the event is a business critical event. For example, you might configure an inquiry as volatile, because an inquiry is not a critical business event and you do not want the system to continually look for the event should the event fail. You might configure a purchase order as reliable, because this is a critical business event and you do want the system to continually look for the event and make the transaction update. Volatile events offer better performance than reliable events, but delivery is not reliable if the event is lost during transportation.

Real-time and XAPI events can be single, aggregate, or composite events. A composite event consists of single events. The composite event and the single events that make up the composite event can have different levels of reliability. For example, you register composite events as RTSOOUT with a level of reliability as reliable, and you register single events as RTSOLINE with a level of reliability as volatile. The level of reliability configured for RTSOOUT will not override the level of reliability that is configured for RTSOLINE. The rationale for this is that the reliability of events is based on the event type. If you decide that single event types are not important enough to configure as reliable delivery, then the single events that are created during composite event creation should have the same level of reliability as other single events.

The APIs you use to create real-time and XAPI events are not affected by the level of reliability.

A.5.2 Configuring Your System for Reliable Event Delivery

To use the reliable event delivery feature, you must define your events in the F90701 table. Use the Interoperability Event Definition (P90701) program to accomplish this task. On the Event Entry form, you must set up the Threshold Timeout field and set the Reliable Delivery field to reliable (either *Y* or *1*). The Timeout Threshold field is in seconds and applies only to the reliable events for which an initial delivery attempt fails. This field determines the maximum amount of time that has to pass from the event creation to the time when the event is going to be discarded if not delivered successfully. Events with a threshold of zero never expire.

Two database tables, the F90704 table and the F90703 table, enable communication between the sender and receiver. Event Protocol stores information that is associated with the protocol that delivers an event. Event Link stores information that is associated with the reliable event for which initial delivery failed. These tables are updated by the system when an event is created.

Note: Both the sender and receiver must access the same instances (the data sources are the same) of the interoperability database tables.

A.5.3 Reliable Event Error Message

If the reliable event is not found, this message might be generated in the client, Callobject, IEO, and EVN logs:

```
RDEL0000045 - Could not open tables for reliable event delivery
(F90703 and F90704). Reliable event delivery will be disabled.
```

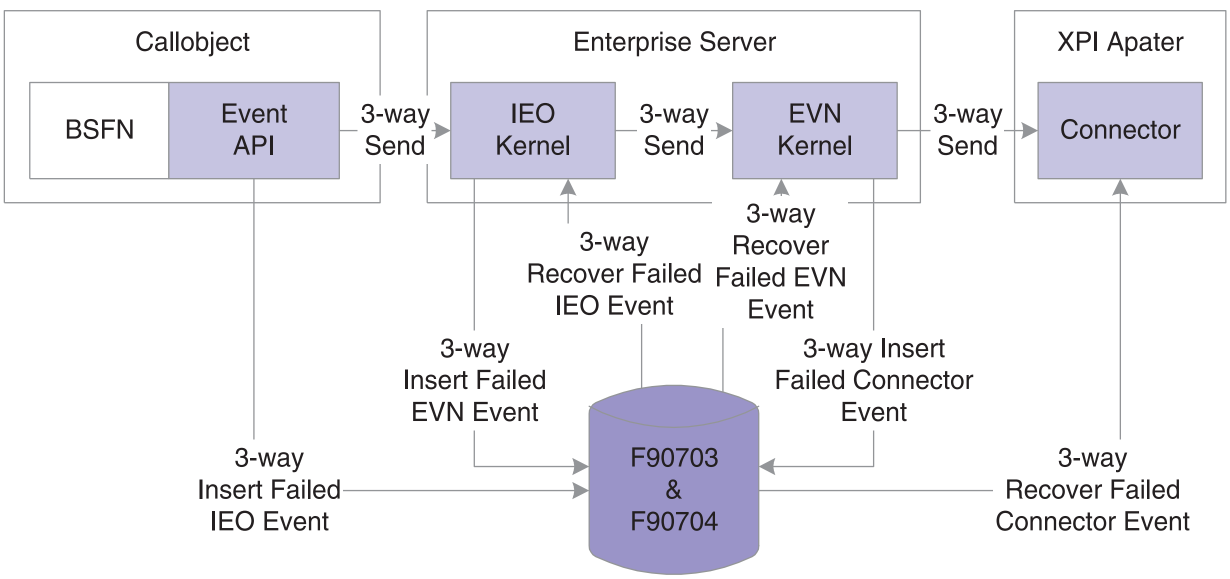
If you receive this error message, verify your events are defined in the F90701 table, that the Reliable Delivery and Threshold Time fields are set up correctly, and that the Event Protocol and Event Link tables exist.

A.5.4 Minimizing Duplicate and Lost Events

The architecture for real-time events processing is changed from a fast request reply (FRR) protocol to a three-way protocol. The three-way protocol enables the storage of event information in the F90703 and F90704 tables. Also, both the Java connector and the COM connector can receive and recover real-time events.

This diagram shows the architecture for real-time event recovery using the three-way protocol:

Figure A–1 Three-way architecture for processing events



If the CallObject kernel is unable to communicate with the IEO kernel, the event API inserts the event information into the F90703 and F90704 tables. The IEO kernel recovers the event information from the tables.

If communication between the IEO and EVN kernels fails, the IEO kernel inserts the information into the F90703 and F90704 tables. The EVN kernel recovers the event information from the F90703 and F90704 tables.

If a communication failure between the EVN kernel and the connector occurs, event information is stored in the F90703 and F90704 tables. Both the Java connector and the COM connector have the ability to recover event information from the F90703 and F90704 tables.

A.5.5 Increasing Performance

To increase performance, the concept of a black list is introduced. The black list is a list of subscribers that are not responding to the event message. The black list concept helps to increase performance by not retransmitting to non-responsive subscribers.

If the EVN kernel cannot send an event to a subscriber, the subscriber is placed on the black list. When a subscriber is placed on the black list, the EVN kernel inserts the event information to the F90703 and F90704 tables until the subscriber is removed from the black list. When the subscriber is removed from the black list, the subscriber receives new event information from the EVN kernel and the connector recovers existing event information from the F90703 and F90704 tables and sends this event information to the subscriber.

Subscribers can be placed on the black list in one of two ways:

- Voluntary
- Forced

A.5.5.1 Voluntary Black List

When a subscriber goes down and sends an unsubscribe message, the EVN kernel adds the subscriber to the black list. No event information is sent to the subscriber until the user re-subscribes. The EVN kernel inserts the event information into the F90703 and F90704 tables, and the information is recovered by the connector once the subscriber re-subscribes. Information about adding and removing the subscriber from the black list can be found in the EVN kernel debug log. These are example EVN kernel debug log messages:

- Added receiver <host name>:<port> to black list.
- Removed receiver <host name>:<port> from black list.

A.5.5.2 Forced Black List

When the EVN kernel sends an event that is defined as reliable to a subscriber, and the subscriber fails to reply to the EVN kernel, the EVN kernel adds that subscriber to the forced black list, and inserts the event information to the F90703 and F90704 tables. Settings that you configure in the jde.ini file determine how many times the EVN kernel sends an event with no response from the subscriber before the subscriber is placed on the black list, and the event information is stored in the database tables. You also configure jde.ini settings that determine how often the system tries to revisit the subscriber to remove that subscriber from the black list.

Information about adding, revisiting, and removing a subscriber can be found in the EVN kernel error log. These are example EVN kernel error log messages:

- Added receiver <host name>:<port> to force black list.
- Revisit receiver <host name>:<port> in force black list.
- Removed receiver <host name>:<port> from force black list.

  More detail information about adding the subscriber to the black list can be found in the EVN kernel debug log. This is an example EVN kernel debug log message:
  ```
  Added receiver <host name>:<port> to forced black list, after
  2 retries with 15 seconds of wait time.
  ```

A.5.6 Configuring the jde.ini File

For reliable event delivery, you must configure these sections and settings in the jde.ini file. These settings are in addition to the settings discussed in the real-time and XAPI events chapters.

A.5.6.1 [INTEROPERABILITY]

Setting	Typical Value	Purpose
EnableBlacklist=	1	A value of 1 enables black list capabilities. The default value is 0 (zero). If you use a value of 0 and your system breaks, your system performance can be affected.
MaxFailedAllowed=	1	Defines the number of failed attempts that the EVN kernel makes to the subscriber before placing the subscriber on the black list. The default value is 3.
ForceBlackListRevisitTime=	60	Defines how often the EVN kernel will attempt to communicate with the failed subscriber once the subscriber is placed on the black list. The default value is 300 seconds.

A.5.6.2 [NETWORK QUEUE SETTINGS]

Setting	Typical Value	Purpose
JDENETTimeout=	60	Defines the time that the EVN kernel waits for a response. **Note:** You should have the same number of JDENET processes as EVN kernels.

A.6 Entering Events

This section provides an overview of entering events in the Interoperability Event Definition table and discusses how to enter single and container events.

A.6.1 Understanding Entering Events

You use the Event Request Definition program (P90701) to add new single and container events and to review your existing events. You add single events by event name. When you add a single event, you must include a data structure. A container event contains either single events, aggregate events, or both. When you add a container event, you define events, single events to be used individually, or data structures, single events to be aggregated. You can change the information for single and container events. You can delete single and container events. You can change the status of an event to active or non-active. If your system has multiple environments, the event status is the same in all environments. You can use menu options to access the subscriber information.

> **Note:** A XAPI event is always a container event, and you must define data structures for XAPI events.

A.6.2 Forms Used to Add Events

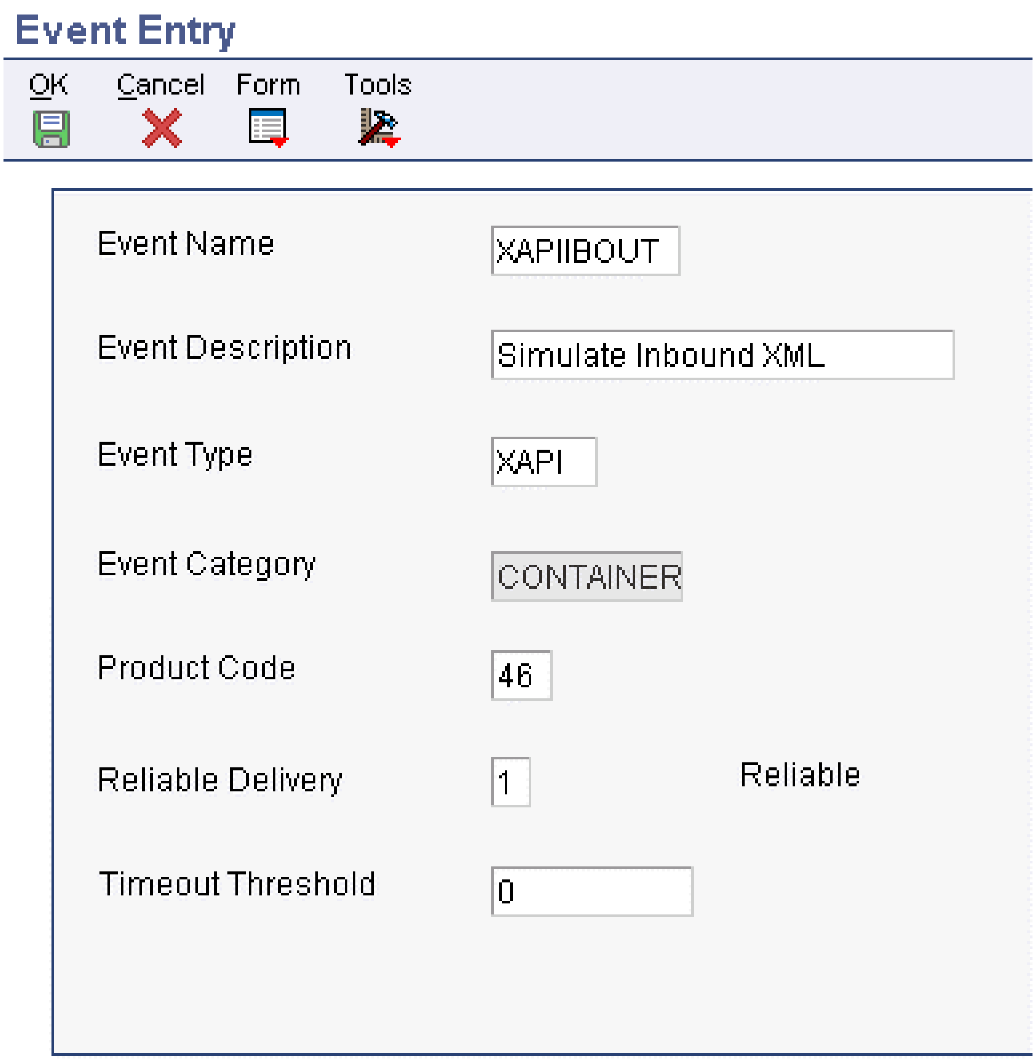
Form Name	FormID	Navigation	Usage
Event Definition Workbench	W90701A	Enter P90701 in the Fast Path.	Locate and review existing single and container events.

Form Name	FormID	Navigation	Usage
Event Entry	W90701D	On Event Definition Workbench, click Add.	Add or change a single or container event.
Event Definition Detail	W90701C	Automatically appears when you click OK on the Event Entry form if you entered Container in the Event Category field for a real-time event or if you entered XAPI in the Event Type field.	Link single events to a container event.

A.6.3 Entering a Single or Container Event

Access the Event Entry form.

Figure A–2 Event Entry form



Event Name
The name of the event (for example, JDERTSOOUT). Single events can be part of other events.

Event Description
The description of an event.

Event Type
A value that represents the name of the event type (for example, the value RTE denotes Real Time Event; the value ZFILE denotes Batch Upload Event).

If you are adding XAPI events, the system automatically completes the Event Category field with Container and after you click OK, the Event Definition Detail form appears. Complete the Data Structure and Data Description fields, and then click OK.

Event Category
The category of the event, for example, single event or event container.

Product Code
A user defined code (98/SY) that identifies a system. Values include:

01: Address Book

03B: Accounts Receivable

04: Accounts Payable

09:: General Accounting

11: Multicurrency

Reliable Delivery
An option that specifies whether the system retransmits and stores failed events. If you clear this option, the system does not retransmit or store failed events. When you select this option, the additional processing might negatively impact system performance. Values are:

1 or Y: Retransmit and store failed events.

0 or N: Do not retransmit or store failed events.

If you are using the Reliable Event Delivery feature, you must set the Reliable Delivery field to reliable (**1** or **Y**) and the Timeout Threshold field must be set.

Timeout Threshold
The Timeout Threshold field is in seconds and applies only to the reliable events for which an initial delivery attempt fails. This field determines the maximum amount of time that has to pass from the event creation to the time when the event will be discarded if not delivered successfully. Events with a threshold of zero never expire.

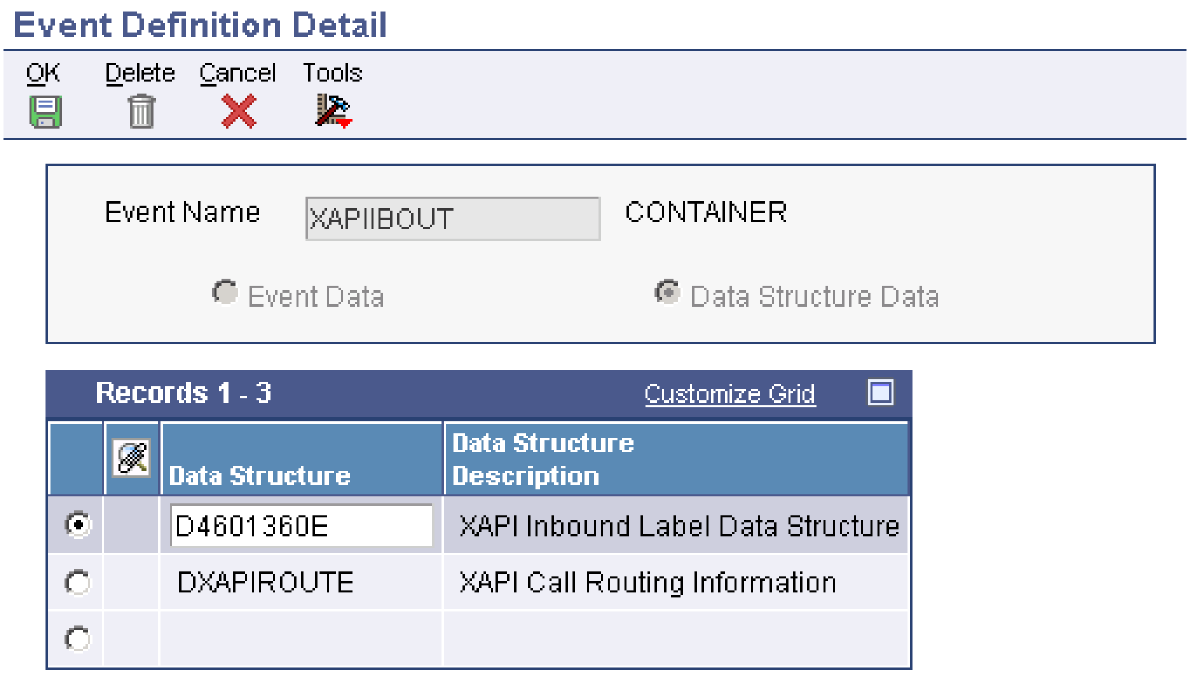
Data Structure
The name of the data structure that passes event information.

This field disappears if Container is the value of the Event Category field; however, when you click OK, the Event Definition Detail form automatically appears for you to enter data structure information.

A.6.3.1 Event Definition Detail

Access the Event Definition Detail form.

Figure A–3 Event Definition Detail form



Event Data
An option that enables you to define single, individual (composite) events for a container event.

Data Structure Data
An option that enables you to define aggregate events for the container event. For XAPI events you must select the Data Structure Data option.

A.6.4 Changing the Status of an Event Record

Access the Event Definition Workbench form.

To change the status of an event:

1. Complete these fields:
   - Event Name
   - Description
   - Event Type
   - Product Code
2. Click the All Statuses option, and then click Find to display existing events.
3. In the detail area, select the event for which you want to change the status.
4. From the Row menu, select Change Status.
5. To view the status change, click Find.

Note: The status of the event is the same for all environments. If the event is active, that event is active for all environments. If the event is non-active, that event is non-active for all environments.

A.7 Adding Logical Subscriber Records

This section provides an overview of the logical subscriber and discusses how to add a logical subscriber record.

A.7.1 Understanding Logical Subscribers

Use the Interoperability Event Subscription program (P90702) to add a logical subscriber. You can also view and modify existing logical subscribers. The Interoperability Event Subscription table contains subscriber information, such as the machine name and port number, and is read by EVN. If subscriber information is missing for the XAPI event, the system generates the event but cannot deliver it.

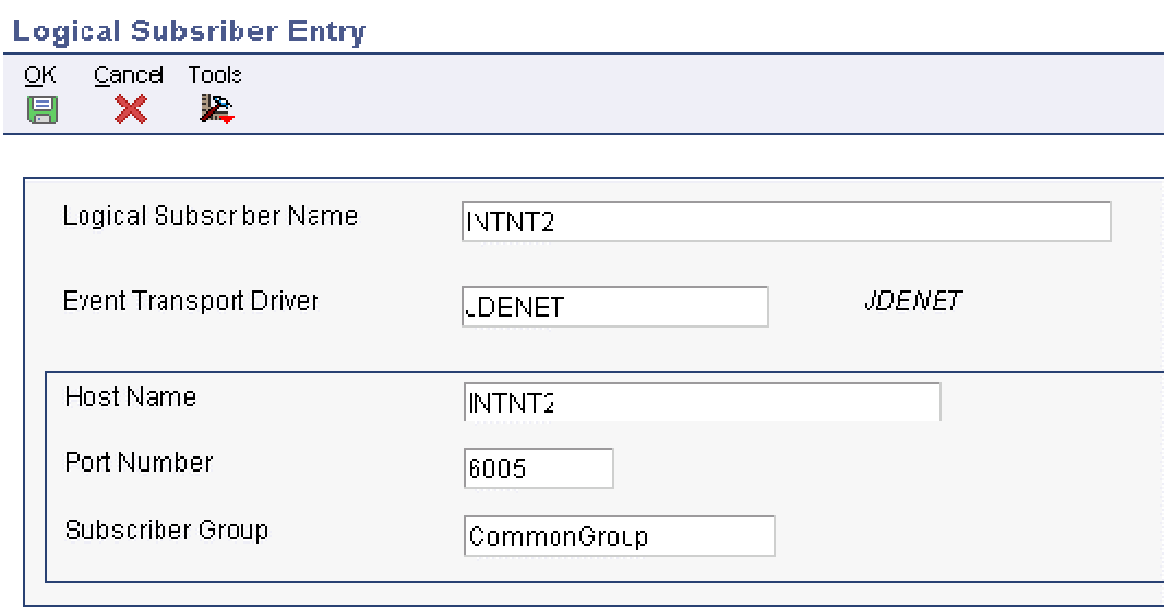
A.7.2 Forms Used to Add a Logical Subscriber

Form Name	FormID	Navigation	Usage
Subscriber Workbench	W90702A	Enter P90702 in the Fast Path.	Locate and review existing subscription information.
Work With Logical Subscriber	W90702D	On Subscriber Workbench, select Logical Subscriber from the Form menu.	Review existing logical subscribers.
Logical Subscriber Entry	W90702B	On Work With Logical Subscriber, click Add.	Add a logical subscriber.

A.7.3 Adding a Logical Subscriber

Access the Logical Subscriber Entry form.

Figure A–4 Logical Subscriber Entry form



Logical Subscriber Name

A value that uniquely identifies a subscriber.

Do not use spaces in the logical subscriber name.

Event Transport Driver
The name of the transport driver that delivers events to the subscriber (for example, JDENET).

Host Name
The name of the server that processes events for the subscriber.

Port Number
A number that identifies the port where the subscriber service is running.

Subscriber Group
A user-defined name that specifies how to deliver events for the subscriber. For example, if you are using a WSG adapter, enter the name of the adapter.

A.8 Entering Subscription Information

This section provides an overview about subscription information and discusses how to:

- Add subscription records.
- Change the status of a subscription record.

A.8.1 Understanding Subscription Records

You use the Interoperability Event Subscription program (P90702) to add new subscription information for XAPI events and to review and change existing subscription information. You can also add a subscription by copying and then modifying an existing subscription, and you can delete subscriptions. You can access and view your real-time and XAPI event definitions by selecting Event Definition from the Form menu. You can also access and view Z events when you click the Z File Events button on the Subscriber Workbench form or by selecting the Z File Events option on the Form menu.

> **Note:** When you add to or modify the F90702 table, you must restart the server for the changes to become effective.

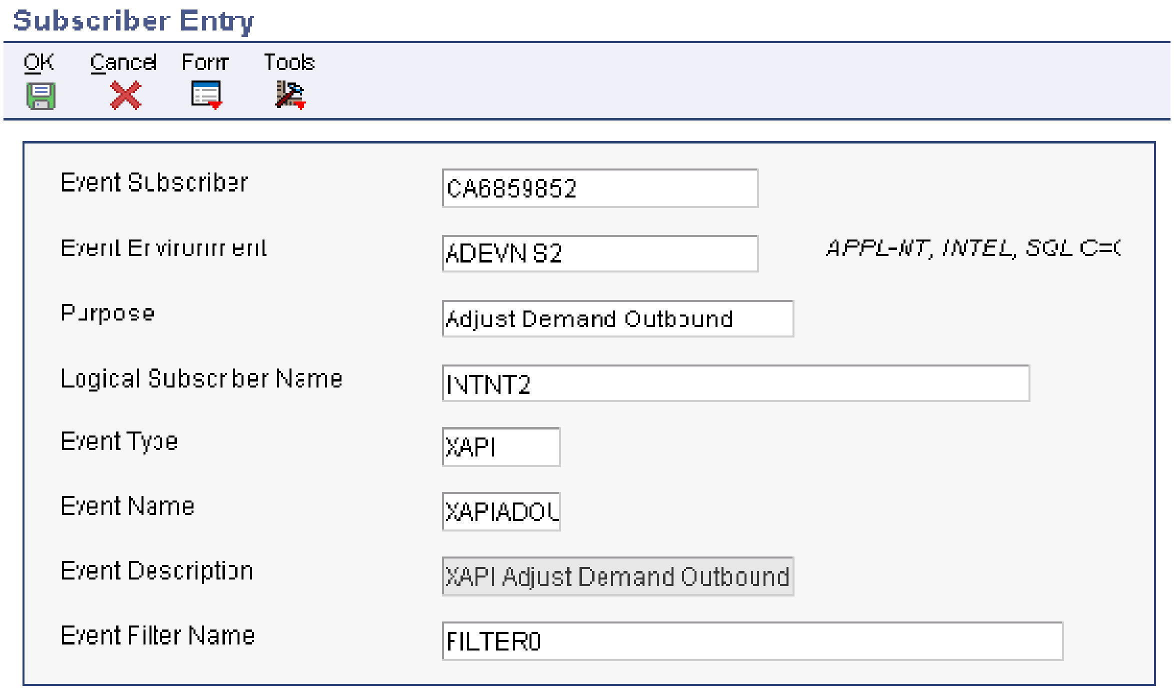
A.8.2 Forms Used to Enter Subscription Information

Form Name	FormID	Navigation	Usage
Subscriber Workbench	W90702A	Enter P90702 in the Fast Path.	Locate and review existing subscription information.
Subscriber Entry	W90702F	On Subscriber Workbench, click Add.	Add a subscription.

A.8.3 Entering a Subscription Record

Access the Subscriber Entry form.

Figure A–5 Subscriber Entry form



Event Subscriber
The user ID for a subscriber.

Event Environment
A value that identifies the environment that the event is executed in.

Purpose
A user defined name or remark.

Logical Subscriber Name
A value that uniquely identifies a subscriber.

Event Type
A value that represents the name of the event type (for example, the value RTE denotes Real Time Event and the value ZFILE denotes Batch Upload Event).

Event Name
The name of the event (for example, JDERTSOOUT). Single events are part of other events.

Event Filter Name
The system automatically enters **Filter0.**

A.8.4 Changing the Status of a Subscription

Access the Subscriber Workbench form.

To change the status of a subscription:

1. Complete these fields:
   – Subscriber Name
   – Purpose

2. Select the All Statuses option, and then click Find to display existing subscriptions.
3. In the detail area, select the event for which you want to change the status.
4. From the Row menu, select Change Status.
5. To view the status change, click Find.

B

Using Classic Real-Time Events

This appendix contains the following topics:

Note: This chapter is applicable only if you use classic events delivery. Classic event delivery is available when you use JD Edwards EnterpriseOne Tools 8.93 or earlier releases, or if you use JD Edwards EnterpriseOne Tools 8.94 with JD Edwards EnterpriseOne Applications 8.10.

Refer to the Guaranteed Events chapters if you use JD Edwards EnterpriseOne Tools 8.94 with JD Edwards EnterpriseOne Applications 8.11 or JD Edwards EnterpriseOne Tools 8.95 and later tools releases with JD Edwards EnterpriseOne Applications 8.10 and later Applications releases.

B.1 Understanding Real-Time Events - Classic

A real-time event is notification that a business transaction has occurred in JD Edwards EnterpriseOne. You can generate real-time events on the JD Edwards EnterpriseOne server using an interface, such as HTML, WIN32, and terminal servers. You can use real-time events for either synchronous and asynchronous processing.

An example of synchronous processing is an auction site that uses JD Edwards EnterpriseOne as a back-end solution that can use real-time events to immediately update the database. For example, a user enters a new item for auction, which triggers a transaction into JD Edwards EnterpriseOne. The system captures the transaction and sends a notification to an interoperability server, such as a Java connector, that communicates the information to a web engine to update HTML pages so that all of the auction users can see the new item.

You can also use real-time event generation for asynchronous processing. For example, an online store sends orders to different vendors (business to business), captures the transactions, and enters the orders into the vendors' systems. A user buys a book. Vendors enter a purchase order to the book publisher and send a notification to the shipping company to pick up the book and deliver it. The book order itself can be completed as a purchase order transaction with JD Edwards EnterpriseOne, but the shipping request requires that the data is packaged into a commonly agreed-upon format for the shipping company to process.

B.1.1 Prerequisites

Before you complete the tasks in this section:

- Enable security for the JD Edwards EnterpriseOne server.
- Ensure that the default user under the [SECURITY] section of the JD Edwards EnterpriseOne server jde.ini file has a valid security record (that is, that the user is a valid JD Edwards EnterpriseOne user).

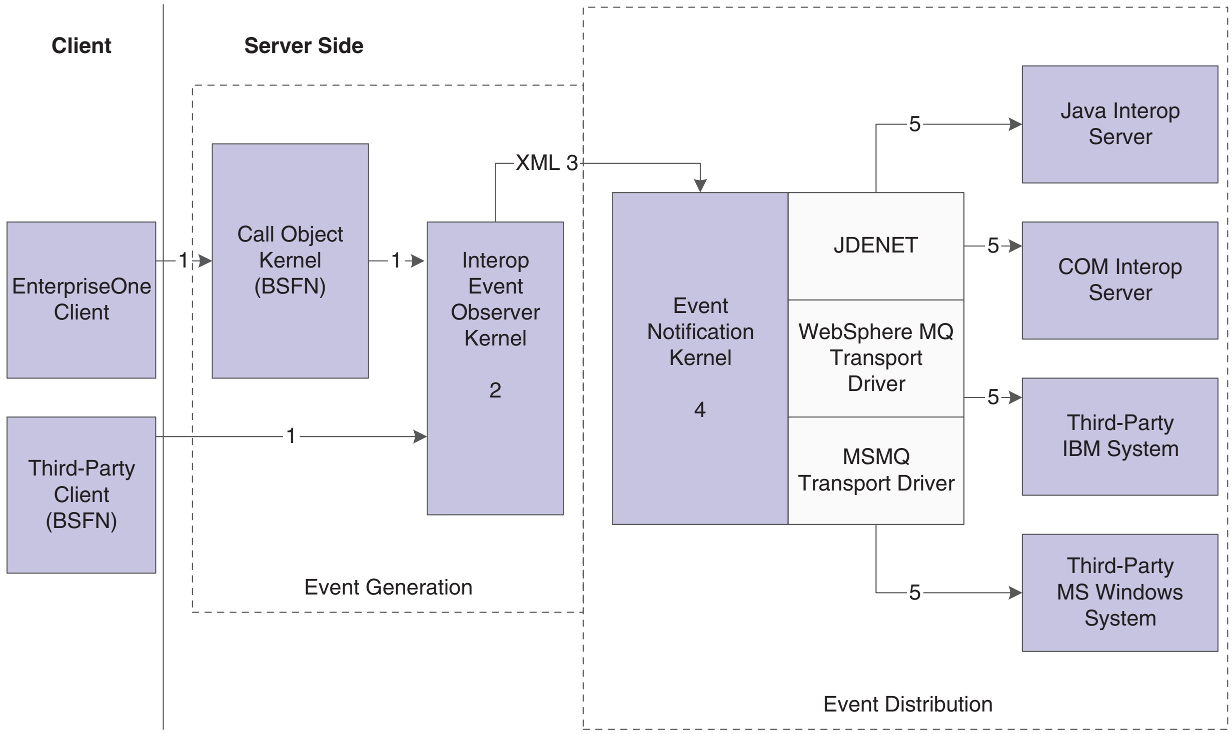
B.2 Processing Real-Time Events

Real-time events use system calls that receive data from business functions that use JD Edwards EnterpriseOne data structures. Each real-time event has a unique ID that includes the JD Edwards EnterpriseOne session ID.

Real-time event generation from a client consists of client business functions that call APIs and interfaces with the server interoperability event observer (a kernel). Real-time event generation on the server side includes an event observer interface (a set of system APIs) that triggers real-time events and an interoperability event observer (a kernel). Then, the event observer kernel generates XML documents of the triggered real-time events and sends them to an event distribution component. The event distribution component is the same one that the system uses to send XML document notification to subscribers.

This diagram is a logical representation of the processes and data for real-time event generation:

Figure B–1 Real-time event process flow



In summary:

1. Event generation is triggered by business functions.

   You use the Object Configuration Manager (OCM) to map business functions to run on the JD Edwards EnterpriseOne server or to run locally. When a business function is mapped to run on the JD Edwards EnterpriseOne server, the business function calls the Interoperability Event Interface within the CallObject kernel. The CallObject kernel sends the information to the Interoperability Event Observation (IEO). When a real-time event is generated from a client, the client business function calls the appropriate API. The API performs OCM lookup to determine where the IEO kernel is located, validates, filters, and formats the data, and then sends the information to the IEO kernel.

2. The IEO kernel creates the real-time event and produces an XML document when the real-time event is finalized.

3. The IEO kernel packages the XML document and passes the document to the Event Notification (EVN) kernel.

4. The EVN kernel determines which transport driver should handle the event.

5. The transport driver distributes the event.

Note: If you use IBM WebSphere MQ or MSMQ transports, the transport drive writes system and function errors to the JDE error log. The driver writes error messages and adds the error codes if available.

B.3 Defining Real-Time Events

You use the Interoperability Event Definition program (P90701) to define real-time events. After you define your real-time event, be sure to activate the event by changing the status to active. If the event is not defined in the F90701 table, the system call returns an error message.

B.4 Using Event Sequencing

When you define your real-time events, you indicate whether the event is reliable or volatile. If you define the event as volatile, the system automatically provides event sequencing to guarantee that events are delivered in the correct order. Volatile events are stamped using JD Edwards EnterpriseOne Next Numbers features.

For sequencing of real-time events, the system call, jdeIEO_EventFinalize, retrieves the event number from the real-time events sequencing bucket, and sends the number to the IEO kernel. The IEO kernel includes the event number as part of the generated event. The IEO kernel sends the event to the EVN kernel. The EVN kernel remembers the last shipped event and bases sequencing on the event number that is contained in the received event.

Event sequencing does impact performance. You can clear events sequencing. You can also define a timeout value to tell the system to stop looking for a missed event when events are out of sequence. The flag and timeout settings are in the [INTEROPERABILITY] section of the jde.ini file.

B.5 Using Journaling

Real-time events are journaled using the trace feature for the JDEDEBUG log files. You can select journaling in the jde.ini file. Journaling occurs in two instances:

- A system call logs the parameter received and the APIs called.
- During the interoperability event observer process, the kernel logs additional debugging information.

  The logging is controlled with the LEVEL key in the [INTEROPERABILITY] section.

B.5.1 [INTEROPERABILITY]

These are some possible values for the LEVEL key:

Key	Section	Description
LEVEL=	N/A	The system writes specified interoperability event data to the debug log file. You can specify one or more of the allowable logging settings. Separate multiple values with a comma. For example: [INTEROPERABILITY] LEVEL=EVENTS,DATA **Note:** As with any logging operation, enabling any of these settings can impact performance and cause extensive amounts of data to be written.
N/A	EVENTS	Use this value to log the flow of events in the IEO kernel. Receiving event data and sending events are logged, but the values of the event data are not logged. This is the default level. If the LEVEL key is not present, it is identical to LEVEL= EVENTS.

Key	Section	Description
N/A	DATA	Log values of the event data and flow of the events in the IEO kernel. This level also includes all data logged with the EVENTS switch.
N/A	PERF	Log statistics about the number of events received and the time period in which they are processed.
N/A	DOC	Outbound XML documents are written in the temporary file on disk. If the debug log is enabled, the document location is written in the debug log. The location of the document depends on the key value: ▪ If the value of the key TempFileDir in the Interoperability section is set, the file is written to that location. For example: ▪ [INTEROPERABILITY] TempFileDir=C:\XML_DOCUMENTS ▪ If the key TempFileDir is not set, files are written in the same directory where JDE logs and debug logs are written. **Note:** Setting the LEVEL=DOC key causes all real-time events to be written to the disk, which can cause a significant performance impact on the JD Edwards EnterpriseOne server. JD Edwards suggests that you not use the LEVEL=DOC setting in a production environment or for stress testing of the quality assurance environment.
N/A	TRACE	This switch traces execution of the IEO kernel and writes data in the debug log. Because of the large amount of data that is logged, use this level only for debugging purposes.

Note: The LEVEL=DOC setting is not affected by whether debug logs are enabled or disabled. All other values under the LEVEL key (for example, TRACE) are affected by the debug log enable or disable setting.

You can also journal EVN documents by setting the SaveEVNDoc key in the [INTEROPERABILITY] section of the jde.ini file. SaveEVNDoc is similar to LEVEL=DOC but applies to the EVN kernel instead of the IEO kernel. The default value for SaveEVNDoc is zero (0), which means that EVN documents are not saved. To save EVN documents, change the value to one (1). EVN documents are saved to the directory where JDE logs and debug logs are written unless you specify a different directory. You can use TempFileDir to specify a directory.

B.5.2 [INTEROPERABILITY]

You can configure these settings to log documents:

```
SaveEVNDoc=1
TempFileDir=C:\XML_Documents
```

B.6 Configuring the jde.ini for Real-Time Events

To generate real-time events, these sections of the JD Edwards EnterpriseOne server jde.ini file must be configured:

- [JDENET_KERNEL_DEF19]
- [JDENET_KERNEL_DEF20]
- [JDEITDRV]
- [JDENET]

- [INTEROPERABILITY]

> **Note:** The settings for the kernels, [JDEITDRV], and [JDENET] are defined in the Using Events - Classic section of the Interoperability Guide.

See Configuring the jde.ini file for Events.

B.6.1 [INTEROPERABILITY]

Configure these settings:

```
SequenceTimeOut=XX
XMLElementSkipNullOrZero=x
```

The SequenceTimeOut setting is for sequencing of volatile events. The default value is 10 seconds.

Null strings and zeros are trimmed from real-time events. You can clear this feature by entering a value of 0 (zero) for the XMLElementSkipNullOrZero settings.

B.7 Generating Real-Time Events

This section provides overview of real-time event generation, real-time event APIs, and example code for events, and discusses how to set up the OCM for real-time events.

B.7.1 Understanding Real-Time Event Generation

Events can be one of these:

- Single event

  Contains one partial event. Single events are useful if the receiver requires that events be generated per system call. Can also be used with different event types.

- Aggregate event

  Contains multiple partial events. Aggregate events are useful if the receiver requires a document that contains multiple events. For example, a supply chain solution might want to provide the complete sales order as one event that contains multiple partial events.

- Composite event

  Contains only single events. Aggregate events are useful if the customer has multiple receivers, some that require single events and some that require a complete event that is similar to an aggregate event.

B.7.2 Real-Time Event APIs

The system APIs are able to determine whether a system call is from a server or client. These APIs are available for you to generate real-time events:

- jdeIEO_EventInit()
- jdeIEO_EventAdd()
- jdeIEO_EventFinalize()

- jdeIEO_CreateSingleEvent()
- jdeIEO_IsEventTypeEnabled()

B.7.3 Example: Interoperability Event Interface Calls

This sample code illustrates how to create a single event:

1. Design the data structure to decide what values to provide to the real-time event.

```
typedef struct tagDSD55RTTEST
   {
   char    szOrderCo[6];
   char    szBusinessUnit[13];
   char    szOrderType[3];
   MATH_NUMERIC   mnOrderNo;
   MATH_NUMERIC   mnLineNo;
   JDEDATE     jdRequestDate;
   char    szItemNo[27];
   char    szDescription1[31];
   MATH_NUMERIC   mnQtyOrdered;
   MATH_NUMERIC   mnUnitPrice;
   MATH_NUMERIC   mnUnitCost;
   char    szUserID[11];
   } DSD55RTTEST, *LPDSD55RTTEST;
```

2. Define the data structure object in the business function header file.
3. Modify the business function source to call jdeIEO_CreateSingleEvent.

```
JDEBFRTN(ID) JDEBFWINAPI RealTimeEventsTest (LPBHVRCOM lpBhvrCom,
   LPVOID lpVoid,
   LPDSD55REALTIME lpDS)
{
/* Define Data Structure Object */
DSD55RTTEST  zRTTest   = {0};
   IEO_EVENT_RETURN  eEventReturn   = eEventCallSuccess;
IEO_EVENT_ID  szEventID   ={0};

()Populate required members

/* Now call the API */

   szEventID = jdeIEO_CreateSingleEvent { lpBhvrCom,
      "RealTimeEventsTest",
      "JDERTOUT",
      "SalesOrder",
      "D55RTTEST",
   &zRTTest,
   sizeof(zRTTest),
   0,
   &eEventReturn   };

   /* Error in jdeFeedCallObjectEvent is not a critical error
          and should only be treated as a warning */
      if( eEventReturn != eEventCallSuccess )
{
      /* LOG the Warning and return */
      return ER_WARNING;
```

This sample code illustrates how to create an aggregate event:

```
DSD55RTTEST zD55TEST01 = {0};
 DSD55RTTEST zD55TEST02 = {0};
 DSD55RTTEST zD55TEST03 = {0};
 IEO_EVENT_RETURN eEventReturn = eEventCallSuccess;
 IEO_EVENT_ID szEventID;

 szEventID = jdeIEO_EventInit (lpBhvrCom, eEventAggregate, "MyFunction1",
 "JDESOOUT", "EventScope1", 0, &eEventReturn);
 eEventReturn = jdeIEO_EventAdd (lpBhvrCom, szEventID, "MyFunction2", NULL,
 "D55TEST01", &zD55TEST01, sizeof(zD55TEST01),0);
 eEventReturn = jdeIEO_EventAdd (lpBhvrCom, szEventID, "MyFunction3", NULL,
 "D55TEST02", &zD55TEST02, sizeof(zD55TEST02),0);
 eEventReturn = jdeIEO_EventAdd (lpBhvrCom, szEventID, "MyFunction3", NULL,
 "D55TEST03", &zD55TEST03, sizeof(zD55TEST03),0);
 eEventReturn = jdeIEO_EventFinalize (lpBhvrCom, szEventID,"MyFunction4",0);
```

This sample code illustrates how to create a composite event:

```
IEO_EVENT_RETURN    eEventReturn    = 0;
    IEO_EVENT_ID szEventID;

   eEventReturn = eEventCallSuccess;
   szEventID = jdeIEO_EventInit (lpBhvrCom, eEventComposite, "MyFunction1",
"JDESOOUT","EventScope1",0,&eEventReturn,0);
   eEventReturn = jdeIEO_EventAdd ( lpBhvrCom, szEventID, "MyFunction2",
"SODOCBEGIN", "D55TEST01", &zD55TEST01, sizeof(zD55TEST01),0);
   eEventReturn = jdeIEO_EventAdd ( lpBhvrCom, szEventID, "MyFunction3",
"SOITEMADD", "EventScope3", "D55TEST02", &zD55TEST02, sizeof(zD55TEST02),0);
   eEventReturn = jdeIEO_EventFinalize (lpBhvrCom, szEventID, "MyFunction4",0);
```

Errors that are returned by the system calls might not be critical enough to stop the business process. The system flags non-critical errors as warnings and logs them in the log file. If the business function is on the server, the warning is logged in the CallObject kernel log. If the business function is on a client, the warning is logged in the client log file.

This sample code illustrates an XML file that shows a composite real-time event that consists of a call to the business function F4211FSEditLine on 12/31/2000, arriving about noon, with the real-time event generated at 12:00:01.000:

```
<?xml version='1.0' encoding='utf-8'  ?>
<jdeResponse type='realTimeEvent' user='JDE1214225' session='1234.786321234'
role='*ALL' environment='XDEVNIS2'>
 <header>
   <eventVersion>1.0</eventVersion>
   <type>JDESOOUT</type>
   <scope>SalesOrder</scope>
   <user>JDE1214225</user>
   <application>P0101</application>
   <version>XJDE101</version>
   <sessionID>1234.786321234</sessionID>
   <environment>XDEVNIS2</environment>
   <host>HP9000B</host>
   <eventID>HP9000B-1234-1231200012000100-JDE1214225-FFA123ECBBA123EC</eventID>
   <date>12312000</date>
   <time>120001000</time>
 </header>

<body elementCount='1'>
   <PartialEvent name='F4211FSEditLine' type='SOEDITLINE' executionOrder='1'
parameterCount='12'>
```

```
 szOrderCo type='String'>JD Edwards</szOrderCo>
     <szBusinessUnit type='String'>Mountain Region</szBusinessUnit>
     <szOrderType type='String'>SO</szOrderType>
     <mnOrderNo type='MATH_NUMERIC'>13209847</mnOrderNo>
     <mnLineNo type='MATH_NUMERIC'>122</mnLineNo>
     <jdRequestedDate type='Date'>12312000</jdRequestedDate>
     <szItemNo type='String'12243234</szItemNo>
     <szDescription type='String'>Bicycle</szDescription>
     <mnQtyOrdered type='MATH_NUMERIC'>1</mnQtyOrdered>
     <mnUnitPrice type='MATH_NUMERIC'>249.99</mnUnitPrice>
     <mnUnitCost type='MATH_NUMERIC'>213.23</mnUnitCost>
     <szUserID type='String'>JDE1214225</szUserID>
   </PartialEvent>
 </body>
</jdeResponse>
```

B.8 Setting Up the OCM for Real-Time Events

This section provides an overview about OCM for real-time events and discusses how to set up the OCM.

B.8.1 Understanding the OCM for Real-Time Events

You configure the Object Configuration Manager (OCM) so that the system call can find the IEO kernel. If the business function is mapped to a client, an error is returned to the client by the system call if the IEO kernel is not found. If the business function is mapped to the server, the error is logged in the Callobject kernel jde.log.

When you configure the OCM, include a specific environment and ensure that no two duplicate mappings are in active status at the same time.

If the OCM mapping is not correctly configured on the client, this message is written in the jde.log, and the event is not be generated:

```
RT0000011 jdeIEO_EventInit: Unable to find the server
```

If the OCM mapping is not correctly configured on the server, no error message is generated. The system call uses the local server as the location of the IEO kernel.

If the IEO kernel is not found on the machine that is configured in the OCM, this error might occur:

```
RT0000004 jdeIEO_EventInit: ReceiveMsg failed. Error = <error test>
```

B.8.2 Forms Used to Set Up OCM

Form Name	FormID	Navigation	Usage
Machine Search and Select	W986110D	Enter **OCM** in the Fast Path Command Line.	Select the appropriate machine name.
Work With Object Mappings	W986110B	On Machine Search and Select, select the appropriate active environment.	Find and select the appropriate environment.

Form Name	FormID	Navigation	Usage
Object Mapping Revisions	W986110C	On Work With Object Mappings, select the appropriate active environment in the detail area.	Enter RTE in the Object Type field.

B.8.3 Setting Up the OCM for Real-Time Events

Access the Object Mapping Revisions form.

Environment Name
A name that uniquely identifies the environment.

Object Name
The JD Edwards EnterpriseOne object that you want to map. To create a default map for all of an object type, enter the literal value **DEFAULT** into this field and then enter an object type into the Object Type field.

Primary Data Source
The name that identifies the data source.

Data sources are the building blocks that you use to set up a JD Edwards EnterpriseOne configuration. Data sources define all of the required databases (where your tables reside) and all of the logic machines (where JD Edwards EnterpriseOne executes logic objects for your enterprise).

If JD Edwards EnterpriseOne cannot find your primary data source or cannot find the data item I your primary data source, it attempts to connect to your secondary data source.

System Role
A profile that classifies users into groups for system security purposes. You use group profiles to give the members of a group access to specific programs.

On this form you can enter an individual user, a group name or the literal value *PUBLIC.

Object Type
The type of object with which you are working. For real-time events, the object type is RTE. For XAPI events the object type is XAPI.

Data Source Mode
Indicates whether to use the primary or secondary data source.

Allow QBE
Use this flag to select row-level record locking for the data source.

You should have this flag turned on to help prevent database integrity issues.

JDEBASE middleware uses this flag to determine whether to use row-level record locking.

C

Using Classic XAPI Events

This appendix contains the following topics:

Note: This chapter is applicable only if you use classic events delivery. Classic event delivery is available when you use JD Edwards EnterpriseOne Tools 8.93 or earlier releases, or if you use JD Edwards EnterpriseOne Tools 8.94 with JD Edwards EnterpriseOne Applications 8.10.

Refer to the Guaranteed Events chapters if you use JD Edwards EnterpriseOne Tools 8.94 with JD Edwards EnterpriseOne Applications 8.11 or JD Edwards EnterpriseOne Tools 8.95 and later tools releases with JD Edwards EnterpriseOne Applications 810 and later Applications releases.

C.1 Understanding XAPI Events - Classic

XAPI is a JD Edwards EnterpriseOne service that captures transactions as the transaction occurs, and then calls third-party software, end-users, and other JD Edwards systems to obtain a return response. A XAPI event is very similar to a real-time event and uses the same infrastructure to send an event. The difference between a real-time event and a XAPI event is that the subscriber to a XAPI event returns a reply to the originator. The XAPI event contains a set of structured data that includes a unique XAPI event name and a business function to be invoked upon return. Like real-time events, XAPI events can be generated on the JD Edwards EnterpriseOne server using any interface, such as HTML, WIN32, and terminal servers.

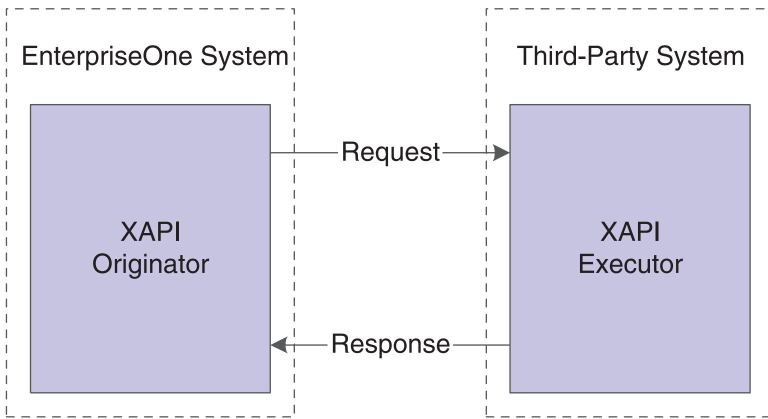
The XAPI structure sends outbound events and receives a reply from third-party systems. An event is generated in JD Edwards EnterpriseOne and sent to a third-party system for processing. The JD Edwards EnterpriseOne system is called the originator. The third-party system sends a response back to JD Edwards EnterpriseOne. The third-party system is called the executor.

The XAPI structure also provides complete request-reply connectivity between two JD Edwards EnterpriseOne systems. The JD Edwards EnterpriseOne system that generates the event is called the Originator. The JD Edwards EnterpriseOne system that responds to the event is called the Executor.

C.1.1 JD Edwards EnterpriseOne to Third-Party

This diagram shows a logical representation of the XAPI process from JD Edwards EnterpriseOne to a third-party system:

Figure C–1 JD Edwards EnterpriseOne to a third-party system XAPI event



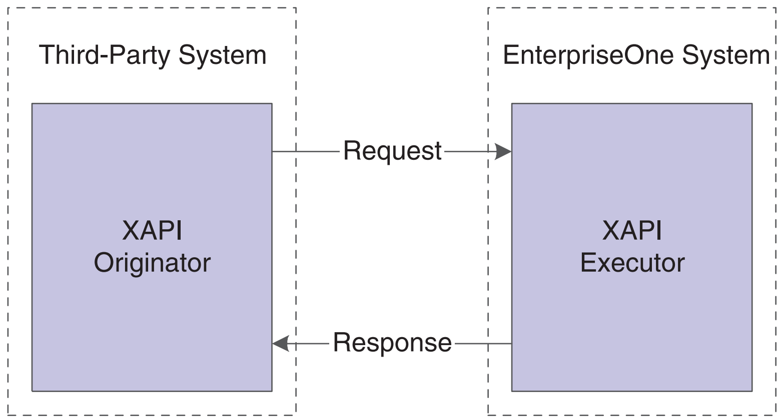
In summary:

1. JD Edwards EnterpriseOne, (XAPI originator) sends a request.
2. The request is sent to a third-party system.
3. The third-party system (XAPI executor) processes the request and sends a response back to the XAPI originator.

C.1.2 Third-Party to JD Edwards EnterpriseOne

This diagram shows a logical representation of the XAPI process from a third-party system to JD Edwards EnterpriseOne:

Figure C–2 Third-party system to JD Edwards EnterpriseOne XAPI event



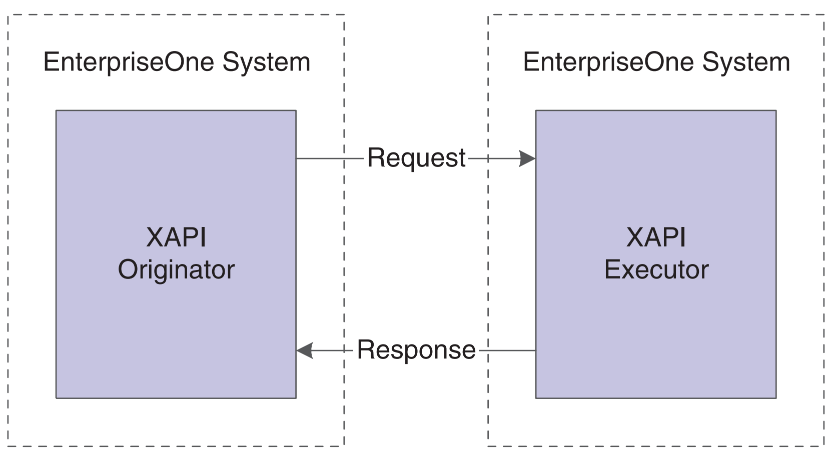
In summary:

1. The third-party system (XAPI originator) sends a request using the JD Edwards EnterpriseOne XAPI request form.
2. The request is sent to JD Edwards EnterpriseOne.
3. JD Edwards EnterpriseOne (XAPI executor) processes the request and sends a response back to the XAPI originator.

C.1.3 JD Edwards EnterpriseOne-to-JD Edwards EnterpriseOne

This diagram shows a logical representation of the XAPI processing for two different JD Edwards EnterpriseOne systems communicating with each other:

Figure C–3 JD Edwards EnterpriseOne-to-EnterpriseOne XAPI event



In summary:

1. The first JD Edwards EnterpriseOne system (XAPI originator) sends a request.
2. The request is sent to a second JD Edwards EnterpriseOne system, which may share the same or different environment as the first JD Edwards EnterpriseOne system.
3. The second JD Edwards EnterpriseOne system (XAPI executor) processes the request and sends a response back to the first JD Edwards EnterpriseOne system (XAPI originator).

4. The first JD Edwards EnterpriseOne system (XAPI originator) processes the response.

C.1.4 Prerequisites

Before you complete the tasks in this section:

- Enable security for the JD Edwards EnterpriseOne server.
- Ensure that the default user has a valid security record under the [SECURITY] section of the JD Edwards EnterpriseOne server jde.ini file (that is, that the user is a valid JD Edwards EnterpriseOne user).

C.2 Defining XAPI Events

You use the Interoperability Event Definition (P90701) program to define XAPI events. When you define XAPI events, the system automatically updates the Event Category field to Container. All XAPI events use the data structure option. The system automatically adds the DXAPIROUTE data structure, which is required for XAPI events. The DXAPIROUTE data structure contains the routing information that is to be returned to the originating system. The jdeXAPI_Finalize API appends DXAPIROUTE data execution. After you define your XAPI event, be sure to activate the event by changing the status.

See Defining Events.

C.3 Subscribing to XAPI Events

If you generate XAPI events, you must define a logical subscriber and set up XAPI event subscriber information. The logical subscriber must exist before you can add XAPI event subscriber information. If subscriber information is missing, the system generates the XAPI event but does not deliver it. You use the Interoperability Event Subscription program (P90702) to define the logical subscriber and to set up XAPI subscriber information. After you set up the XAPI subscriber, be sure to activate the subscriber by changing the status.

See Subscribing to Events.

C.4 Setting Up the OCM for XAPI Events

If your interface to JD Edwards EnterpriseOne is not a JD Edwards EnterpriseOne client, you must configure the OCM so that the system call can find the IEO kernel. When you configure the OCM, include a specific environment and ensure that no two duplicate mappings are in active status at the same time.

To configure the OCM, access the Object Mapping Revisions form and enter XAPI in the Object Type field. Configuring the OCM with the XAPI entry enables the system call to find the IEO kernel. If the OCM is not properly configured, the system generates an error message. OCM error messages for XAPI events are the same as the OCM error messages for real-time events.

See Understanding the OCM for Real-Time Events.

C.5 Working with JD Edwards EnterpriseOne and Third-Party XAPI Events

This section provides an overview of the XAPI event generation and response and discusses:

- XAPI outbound request process flow.
- XAPI outbound request APIs.
- XAPI outbound request API usage sample code.
- XAPI outbound request XML sample code.
- XAPI outbound request jde.ini file configuration.
- XAPI inbound response process flow.
- XAPI inbound response parsing APIs.
- XAPI inbound response parsing API usage sample code.
- XAPI inbound response sample code.
- XAPI inbound response jde.ini file configuration.
- XAPI client jde.ini file configuration.

C.5.1 Understanding XAPI Event Generation and Third-Party Response

The XAPI structure supports XAPI outbound event generation. XAPI outbound events are generated by the XAPI originator exactly the same as real-time events.

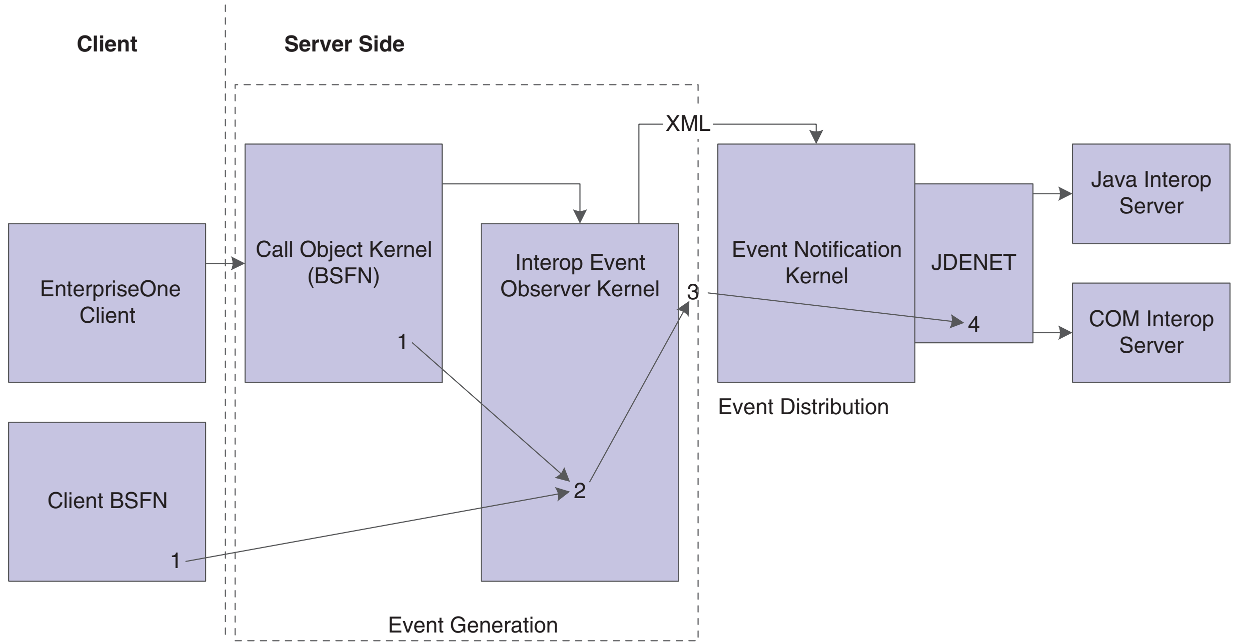
The XAPI structure also provides for an inbound response. The XAPI inbound response happens after a XAPI event is generated. The XAPI inbound response is handled by the third-party system. The third-party system, the XAPI executor, processes the request (event) and returns a reply to the XAPI originator.

When the return XML document is received, it is routed to the XML Service kernel. The XML Service kernel saves the XML document to disk, creates a unique handle, and then calls the callback business function that is provided in the DXAPIROUTE XAPI method ID element in the XML document.

C.5.2 XAPI Outbound Request Process Flow

This diagram illustrates the flow for a XAPI outbound request that is sent to a third-party system:

Figure C–4 XAPI request to a third-party system



In summary:

1. When a XAPI event is generated from a JD Edwards EnterpriseOne client, the client business function calls the appropriate API.

   This API does an OCM lookup to determine where the IEO kernel is located. The API validates, filters, and formats the data. When a XAPI event is generated from a JD Edwards EnterpriseOne server, the business function calls the interoperability event interface within the CallObject kernel. The data is sent as a partial event to the IEO kernel.

2. The IEO kernel creates the XAPI event and produces an XML document when the XAPI event is finalized.
3. The IEO kernel packages the XML document and passes the document to the EVN kernel.
4. The EVN kernel determines the transport driver that should handle the event, and JDENET distributes the information to the subscribers.

Note: XAPI currently does not use IBM WebSphere MQ or MSMQ. All events that are defined in the F90701 table are sent to you if you configure your system to receive events using IBM WebSphere MQ and MSMQ transport drivers.

C.5.3 XAPI Outbound Request APIs

These APIs are available for you to generate a XAPI call:

- jdeXAPI_Init
- jdeXAPI_Add
- jdeXAPI_Finalize
- jdeXAPI_Free

- jdeXAPI_SimpleSend
- jdeXAPI_ISCallTypeEnabled
- jdeXAPI_CALLS_ENABLED

C.5.4 XAPI Outbound Request API Usage Sample Code

This code sample illustrates how to create a XAPI outbound request:

```
/* Header files required */

#include <B4205010.h>

/************************/
 BOOL bXAPIInUse, bExit;
#ifdef jdeXAPI_CALLS_ENABLED
 XAPI_CALL_ID ulXAPICallID = 0;
 XAPI_CALL_RETURN eXAPICallReturn = eEventCallSuccess;
#endif
 DSD4205010A dsD4205010A = {0}; /*Query Header*/
 DSD4205010B dsD4205010B = {0}; /*Query Detail*/
#ifdef jdeXAPI_CALLS_ENABLED
 if(jdeXAPI_IsCallTypeEnabled("XAPIOPOUT") && jdeXAPI_IsCallTypeEnabled⇒
("XAPIOPIN") )
 {
  bXAPIInUse = TRUE;
 }
#endif
 /*-----------------------------------------------------*/
 /* Call XAPIInit */
#ifdef jdeXAPI_CALLS_ENABLED
 if(bXAPIInUse == TRUE)
 {
  ulXAPICallID = jdeXAPI_Init( lpBhvrCom, "SendOrderPromiseRequest",
"XAPIOPOUT", NULL, &eXAPICallReturn);
  if (eXAPICallReturn != eEventCallSuccess)
  {
   bExit = TRUE;
  }
 }
#endif
 /*-------------------------------------------------*/
 /* Adding Header Information */
#ifdef jdeXAPI_CALLS_ENABLED
 if(bXAPIInUse == TRUE)
 {
  eXAPICallReturn = jdeXAPI_Add( lpBhvrCom, ulXAPICallID,
"SendOrderPromiseRequest", "D4205010A", &dsD4205010A, sizeof(DSD4205010A));
  if (eXAPICallReturn != eEventCallSuccess)
  {
   bExit = TRUE;
  }
 }
#endif
 /*-------------------------------------------------*/
 /* Loading Detail Information */
#ifdef jdeXAPI_CALLS_ENABLED
 if(bXAPIInUse == TRUE)
 {
  eXAPICallReturn = jdeXAPI_Add( lpBhvrCom, ulXAPICallID,
```

```
"SendOrderPromiseRequest", "D4205010B", &dsD4205010B, sizeof(DSD4205010B));
  if (eXAPICallReturn != eEventCallSuccess)
  {
   bExit = TRUE;
  }
 }
#endif
#ifdef jdeXAPI_CALLS_ENABLED
 if(bXAPIInUse == TRUE)
 /*-------------------------------------------------*/
 /* Finalize */
 {
  eXAPICallReturn = jdeXAPI_Finalize( lpBhvrCom, ulXAPICallID,
"SendOrderPromiseRequest", "OrderPromiseCallback");
  if (eXAPICallReturn != eEventCallSuccess)
  {
   bExit = TRUE;
  }
 }
#endif
#ifdef jdeXAPI_CALLS_ENABLED
 if (eXAPICallReturn != eEventCallSuccess)
 {
 /*-------------------------------------------------*/
 /* CleanUp */
  if(bXAPIInUse == TRUE)
  {
   jdeXAPI_Free( lpBhvrCom, ulXAPICallID, "SendOrderPromiseRequest");
  }
 }
#endif
```

C.5.5 XAPI Outbound Request XML Sample Code

This code example shows the XML template for a XAPI outbound request:

```
xml version=1.0 encoding="utf-8" ?>
<jdeResponse type="realTimeEvent" user="KL5449350" role='*ALL'
session="22558100.1004460662" subtype="XAPICall" environment="DV7333">
<event>
<header>
<eventVersion>1.0</eventVersion>
<type>XAPIOPOUT</type>
<user>KL5449350</user>
<application>APIDRV</application>
<version />
<sessionID>22558100.1004460662</sessionID>
<environment>DV7333</environment>
<host>DEN-PP6954083</host>
<sequenceID>DEN-PP6954083_1540_10302001095648_KL5449350_1</sequenceID>
<date>10302001</date>
<time>095649</time>
<scope />
<codepage>utf-8</codepage>
</header>
<body elementCount="3">
<detail date="10302001" name="APIDRVFunction" time="9:56:48" type= ""
DSTMPL="D4205010A" executionOrder="1" parameterCount="23">
<szRequestId type="String">1234567</szRequestId>
<szUserId type="String">TestUser</szUserId>
```

```
<szQueryMode type="String">Test</szQueryMode>
<szCustomerName type="String">John Doe</szCustomerName>
<mnCustomerId type="Double">12345</mnCustomerId>
<szCustomerGroup type="String">Group 1</szCustomerGroup>
<szAddress1 type="String">Line 1</szAddress1>
<szAddress2 type="String">Suite 1</szAddress2>
<szAddress3 type="String">123 E. Main</szAddress3>
<szPostalCode type="String">50001</szPostalCode>
<szCity type="String">Centennial</szCity>
<szCounty type="String">Arap</szCounty>
<szStateProvince type="String">CO</szStateProvince>
<szCountry type="String">US</szCountry>
<szBusinessObjective type="String" />
<mnTraceDepth type="Double">0</mnTraceDepth>
<mnPenaltyCostAdjustment type="Double">0</mnPenaltyCostAdjustment>
<szOrderNumber type="String">1000</szOrderNumber>
<nAllowBackorders type="Int">49</nAllowBackorders>
<nAllowSubstitution type="Int">48</nAllowSubstitution>
<nAllowPartialLineShip type="Int">49</nAllowPartialLineShip>
<nAllowPartialOrderShip type="Int">49</nAllowPartialOrderShip>
<nAllowMultisource type="Int">49</nAllowMultisource>
</detail>
<detail date="10302001" name="APIDRVFunction" time="9:56:49" type= ""
DSTMPL="D4205010B" executionOrder="2" parameterCount="17">
<mnLineNumber type="Double">1</mnLineNumber>
<mnCacheLineNumber type="Double">1</mnCacheLineNumber>
<mnItemNumber type="Double">2222</mnItemNumber>
<sz2ndItemNumber type="String">1234567</sz2ndItemNumber>
<sz3rdItemNumber type="String">2234567</sz3rdItemNumber>
<szOrderUnit type="String">123</szOrderUnit>
<mnOrderQuantity type="Double">12</mnOrderQuantity>
<szPlanningUnit type="String">ECL</szPlanningUnit>
<mnPlanningQuantity type="Double">12</mnPlanningQuantity>
<mnPlanningMultiple type="Double">1</mnPlanningMultiple>
<mnPlanningUnitPrice type="Double">1234</mnPlanningUnitPrice>
<jdRequestDate type="Date">10302001</jdRequestDate>
<szShippingGroup type="String">Ship Group</szShippingGroup>
<szMultiSource type="String">MS</szMultiSource>
<nAllowPartialLineShip type="Int">49</nAllowPartialLineShip>
<nAllowBackorders type="Int">49</nAllowBackorders>
<nAllowSubstitution type="Int">48</nAllowSubstitution>
</detail>

/* DXAPIROUTE Routing Information */
<detail date="10302001" name="XAPICall time="09:56:49" type=""
DSTMPL="DXAPIROUTE"
executionOrder="3" parameterCount="4">
<ClientPort type="Int">6009</ClientPort>
<ClientIP type="Int">167810863</ClientIP>
<ClientMagicNumber type="Int">32781408</ClientMagicNumber>
<XAPIMethodID type="String">GetComputerID</XAPIMethodID>
</detail>
/* End of DXAPIROUT Routing Information */

</body>
</event>
</jdeResponse>
```

C.5.5.1 Routing Information

All XAPI events must include DXAPIROUTE in the XML file, as noted near the end of the XML code sample. DXAPIROUTE contains the routing information that is to be returned to the originating client. The jdeXAPI_Finalize API appends DXAPIROUTE data execution.

C.5.6 XAPI Outbound Request jde.ini File Configuration

To generate XAPI events, these sections of the JD Edwards EnterpriseOne server jde.ini file must be configured:

- [JDENET_KERNEL_DEF19]
- [JDENET_KERNEL_DEF20]
- [JDEITDRV]

If the jde.ini file is not properly configured for XAPI events, this error message is written to the jde.log file:

```
XAPI Event [Event Name] cannot be subscribed. Must have XAPI Definition in the
INI file.
```

Make sure the XAPI event is defined in the F90701 table and that XAPI Executor information is defined in the jde.ini file.

You can ignore this error message because XAPI subscription is persisted and cannot be unsubscribed:

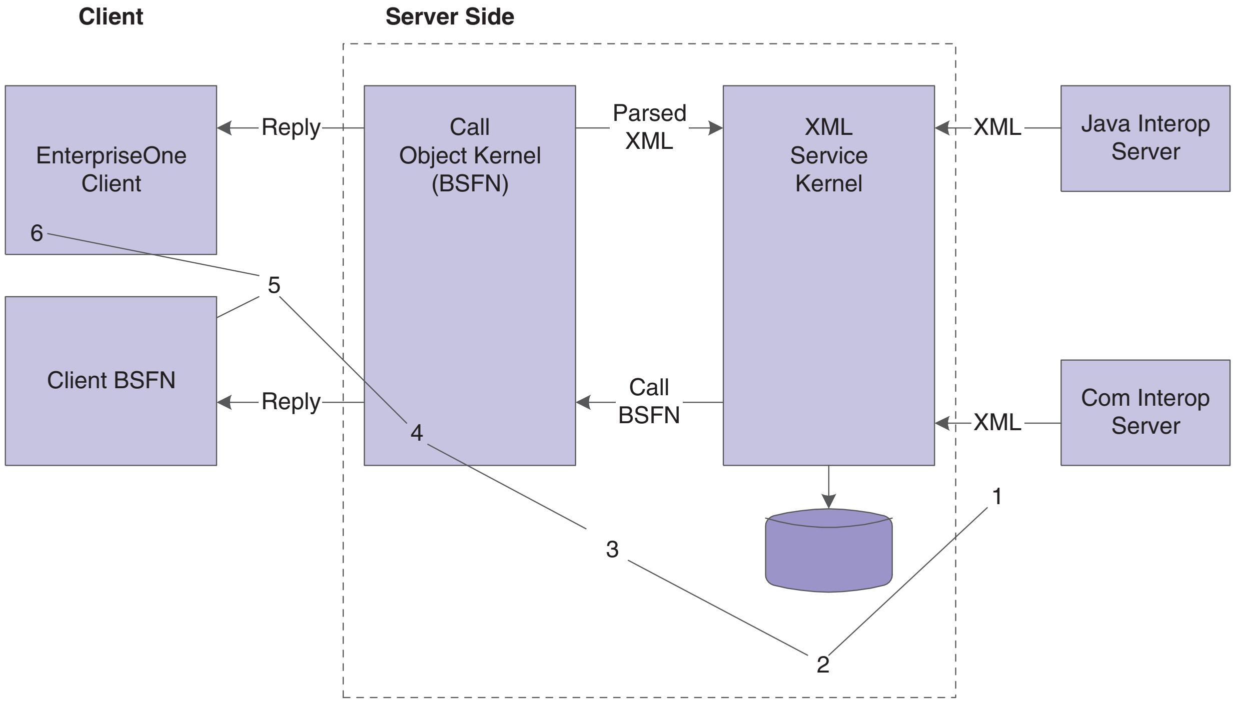
```
Cannot unsubscribe XAPI event.
```

See Configuring the jde.ini file for Events.

C.5.7 XAPI Inbound Response Process Flow

This diagram illustrates the flow for a XAPI inbound response from a third-party system to the JD Edwards EnterpriseOne originating system:

Figure C–5 XAPI response from a third-party system



In summary:

1. An inbound XML document is passed from a third-party system to the XML Service kernel.
2. The XML Service kernel creates a unique XML handle and stores the document on disk.
3. The XML Service kernel reads the XAPICallMethod attribute from the XML document and passes the XML handle as the parameter to the specified business function.
4. The business function (XAPICallMethod) uses XML service APIs to read and parse the XML data into JD Edwards EnterpriseOne data.
5. The business function (XAPICallMethod) uses XML CallObject to send the reply to the originator.
6. A JD Edwards EnterpriseOne client can poll for the XAPI response from the JD Edwards EnterpriseOne server.

C.5.8 XAPI Inbound Response Parsing APIs

These APIs are available for you to generate an inbound XAPI response:

- jdeXML_GetDSCount
- jdeXML_GetDSName
- jdeXML_ParseDS
- jdeXML_DeleteXML

C.5.9 XAPI Inbound Response Parsing API Usage Sample Code

This code example illustrates how the business function uses the XML service APIs to read and parse the XML data:

```
#include <B4205030.h>

 int iCurrentRecord;
 int iHeaderCount;
 int iRecordCount;
 NID nidDSName;
 DSD4205030A dsD4205030A = {0};
 DSD4205030B dsD4205030B = {0};
#ifdef jdeXAPI_CALLS_ENABLED
 if(jdeXAPI_IsCallTypeEnabled("XAPIOPOUT") && jdeXAPI_IsCallTypeEnabled
("XAPIOPIN") )
 {
  iRecordCount = jdeXML_GetDSCount(lpDS->szXMLHandle);
  if (iRecordCount > 0)
  {
   for (iCurrentRecord = 0; iCurrentRecord < iRecordCount; iCurrentRecord++)
   {
    jdeXML_GetDSName(lpDS->szXMLHandle,iCurrentRecord,nidDSName);
    if (jdestrcmp(nidDSName,(const char*)"D4205030A") == 0)//mod
    {
     jdeXML_ParseDS( lpDS->szXMLHandle,iCurrentRecord,&dsD4205030A,
sizeof(DSD4205030A));

    }
    else
    {
     jdeXML_ParseDS( lpDS->szXMLHandle,iCurrentRecord,&dsD4205030B,
sizeof(DSD4205030B));
    }
   }
  }
  if (iCurrentRecord == iRecordCount)
  {
   jdeXML_DeleteXML(lpDS->szXMLHandle);
  }
 }
#endif
```

C.5.10 XAPI Inbound Response Sample Code

This sample code shows an inbound XAPI response:

```
<?xml version="1.0" encoding="utf-8" ?>
<jdeRequest pwd="JDE" type="xapicallmethod" user="JDE" role='*ALL'
session= "" ⇒
environment="DV7333" sessionidle="">
<header>
<eventVersion>1.0</eventVersion>
<type>XAPIOPIN</type>
<user>JDE</user>
<application>XPI</application>
<version />
<sessionID />
<environment>DEVXPINT</environment>
<host>denxpi7</host>
<sequenceID />
```

```
<date>09122001</date>
<time>094951</time>
<scope />
<codepage>utf-8</codepage>
</header>
<body elementCount="3">
<params type="D4205030A" executionOrder="1" parameterCount="24">
<param name="type" />
<param name="dateStamp" />
<param name="timeStamp" />
<param name="szRequestId">1|ZJDE0001</param>
<param name="szBusinessObjective">Maximize_Service</param>
<param name="mnResultNumber">0.0</param>
<param name="mnTotalCost">0.0</param>
<param name="mnTotalDeliveryCost">0.0</param>
<param name="mnTotalPrice">0.0</param>
<param name="mnTotalProfit">0.0</param>
<param name="mnTotalMargin">0.0</param>
<param name="mnTotalValue">0.0</param>
<param name="mnLatestLineDate">0.0</param>
<param name="mnNumberOfBackorders">0.0</param>
<param name="mnNumberOfSubstitutions">0.0</param>
<param name="mnOrderFillRate">0.0</param>
<param name="szErrorCode" />
<param name="szErrorDescription" />
<param name="szOrderNumber">3115|SO|00200</param>
<param name="nAllowPartialOrderShip">0</param>
<param name="nAllowMultisource">0</param>
<param name="nAllowBackorders">0</param>
<param name="nAllowSubstitution">0</param>
<param name="nAllowPartialLineShip">0</param>
</params>
<params type="D4205030B" executionOrder="2" parameterCount="28">
<param name="type" />
<param name="dateStamp" />
<param name="timeStamp" />
<param name="mnLineNumber">1.0</param>
<param name="mnOriginalLineNumber">1.0</param>
<param name="mnCacheLineNumber">1.0</param>
<param name="mnRequestedItem">60011.0</param>
<param name="mnAvailableItem">60011.0</param>
<param name="mnAvailableAmount">25.0</param>
<param name="jdAvailableDate">09/12/2001 00:00:00</param>
<param name="jdRequestedDate">09/10/2001 00:00:00</param>
<param name="jdPickDate">09/11/2001 00:00:00</param>
<param name="jdShipDate">09/11/2001 00:00:00</param>
<param name="szShipLocation" />
<param name="mnCost">0.0</param>
<param name="mnDeliveryCost">0.0</param>
<param name="mnPrice">0.0</param>
<param name="mnProfit">0.0</param>
<param name="mnMargin">0.0</param>
<param name="mnValue">0.0</param>
<param name="mnSubstitutionRatio">0.0</param>
<param name="szShippingGroup" />
<param name="szMultiSource" />
<param name="szErrorCode" />
<param name="szSuspectedCause" />
<param name="nAllowPartialOrderShip">0</param>
<param name="nAllowBackorders">0</param>
```

```
<param name="nAllowSubstitution">0</param>
</params>
<params type="DXAPIROUTE" executionOrder="3" parameterCount="7">
<param name="type" />
<param name="dateStamp">09/05/2001 00:00:00</param>
<param name="timeStamp">13:54:04</param>
<param name="ClientPort">6009</param>
<param name="ClientIP">168045665</param>
<param name="ClientMagicNumber">3</param>
<param name="XAPIMethodID">OrderPromiseCallback</param>
</params>
</body>
</jdeRequest>
```

C.5.11 XAPI Inbound Response jde.ini File Configuration

These sections of the JD Edwards EnterpriseOne server jde.ini file must be configured for the XAPI response portion of the XAPI structure:

- [JDENET_KERNEL_DEF22]
- [JDENET_KERNEL_DEF24]
- [XAPI]
- [XMLLookupInfo]

C.5.11.1 [XAPI]

Configure this setting:

```
XMLDirectory=c:\builds\bdev\log\
```

Note: The XML document directory (XMLDirectory) must be registered in the jde.ini file on the server under the [XAPI] section in the XMLDirectory key. The key contains the directory on the server where XML documents are to be stored.

Caution: For data privacy, be sure to remove the global read access rights for the specified directory.

C.5.11.2 [XMLLookupInfo]

Configure these settings:

```
XMLRequestType5=realTimeEvent
XMLKernelMessageRange5=14251
XMLKernelHostName5=local
XMLKernelPort5=0
XMLKernelReply5=0
```

See Understanding the OCM for Real-Time Events.

C.5.12 XAPI Client jde.ini File Configuration

If you are using a JD Edwards EnterpriseOne client to generate XAPI events, you must define the Client Dispatch kernel and [JDENET] sections of the client jde.ini file. If your interface to the JD Edwards EnterpriseOne server is other than a JD Edwards

EnterpriseOne client, these two settings are not required. The settings enable the JD Edwards EnterpriseOne client to poll for the XAPI response message from the JD Edwards EnterpriseOne server.

Use these settings to configure your JD Edwards EnterpriseOne client jde.ini file.

C.5.12.1 [JDENET_KERNEL_DEF27]

Configure these settings:

```
krnlName=CLIENT DISPATCH KERNEL
dispatchDLLName=jdeuser.dll
dispatchDLLFunction=_JDENET_ClientDispatch
maxNumberOfProcesses=0
numberOfAutoStartProcesses=0
```

C.5.12.2 [JDENET]

Configure these settings

```
serviceNameListen=6004
serviceNameConnect=6004
maxKernelRanges=27
netTrace=0
```

Note: The serviceNameListen and serviceNameConnect settings must be the same as the server's settings. For example, if your server jde.ini file has serviceNameListen=6005 and serviceNameConnect=6005, then your JD Edwards EnterpriseOne client jde.ini file must be serviceNameListen=6005 and serviceNameConnect=6005.

The value for maxKernelRanges setting should be the same value as the server.

C.6 Working with JD Edwards EnterpriseOne-to-EnterpriseOne XAPI Events

This section provides an overview of the JD Edwards EnterpriseOne-to-EnterpriseOne XAPI events and discusses:

- XAPI JD Edwards EnterpriseOne-to-EnterpriseOne process flow.
- XAPI outbound request generation APIs.
- XAPI outbound request handling APIs.
- XAPI outbound request parsing API usage sample code.
- XAPI JD Edwards EnterpriseOne originator XML sample code.
- XAPI inbound response generation APIs.
- XAPI inbound response parsing API usage sample code.
- XAPI response from originator system sample code.
- XAPI inbound response handling APIs.
- XAPI error handling APIs.
- XAPI JD Edwards EnterpriseOne-to-EnterpriseOne jde.ini file configuration.

C.6.1 Understanding JD Edwards EnterpriseOne-to-EnterpriseOne XAPI Events

The XAPI structure provides the capability for two different JD Edwards EnterpriseOne systems to communicate with each other. The first JD Edwards EnterpriseOne system (XAPI originator system) generates a XAPI request (event). Instead of the request being distributed to a third-party system, JDENET sends the request to a second JD Edwards EnterpriseOne system.

You can use the reliable event delivery feature to process XAPI events.

C.6.1.1 Modifying Element Name for XML Documents

Before XAPI event processing, any document that was sent from JD Edwards EnterpriseOne was considered to be a response document, and any document coming in to JD Edwards EnterpriseOne was considered to be a request document. However, with XAPI, request documents are generated by the JD Edwards EnterpriseOne originating system and can be sent to a JD Edwards EnterpriseOne executor system. Response documents are generated and sent by the JD Edwards EnterpriseOne executor system and received by the JD Edwards EnterpriseOne originating system. To support XAPI and to enable the XML dispatch kernel to distinguish between a response and reply, JD Edwards created these type attributes to be used with the jdeResponse element:

Attribute Type	Explanation
jdeResponse=RealTimeEvent	Use this element and attribute to identify a XAPI request from the JD Edwards EnterpriseOne originating system and sent to the JD Edwards EnterpriseOne executor system.
jdeResponse=xapicallmethod	Use this element and attribute to identify a XAPI response from the JD Edwards EnterpriseOne executor system and sent to the JD Edwards EnterpriseOne originating system.

When the XMLDispatch kernel receives a document with the jdeResponse element and a RealTimeEvent or xapicallmethod type attribute, XMLDispatch sends the document to the XML Service kernel. XML Service can distinguish a response or a reply based on the type attribute that is associated with the jdeResponse element and then processes the document appropriately.

C.6.1.2 Security for Originator and Executor

Access to the JD Edwards EnterpriseOne originator and JD Edwards EnterpriseOne executor systems is based on:

- User ID
- Password
- Environment
- Role

The JD Edwards EnterpriseOne originating system verifies that the security information is valid and creates an hUser object with an encrypted password to send to the JD Edwards EnterpriseOne executor. Encryption APIs (jdeEnchyper and jdeDecypher) are used to encrypt and decode the password. The security information is sent in the XAPI request XML document.

Note: The user ID, password, environment, and role must be the same on both JD Edwards EnterpriseOne systems (originator and executor).

C.6.1.3 Error Processing for Originator and Executor

You might encounter these two types of errors during XAPI error processing between two JD Edwards EnterpriseOne systems:

Type of Error	Explanation
Business-related errors	The business function or the business function specifications cannot be found.
System errors	These errors occur in other parts of the system (for example, message delivery failure).

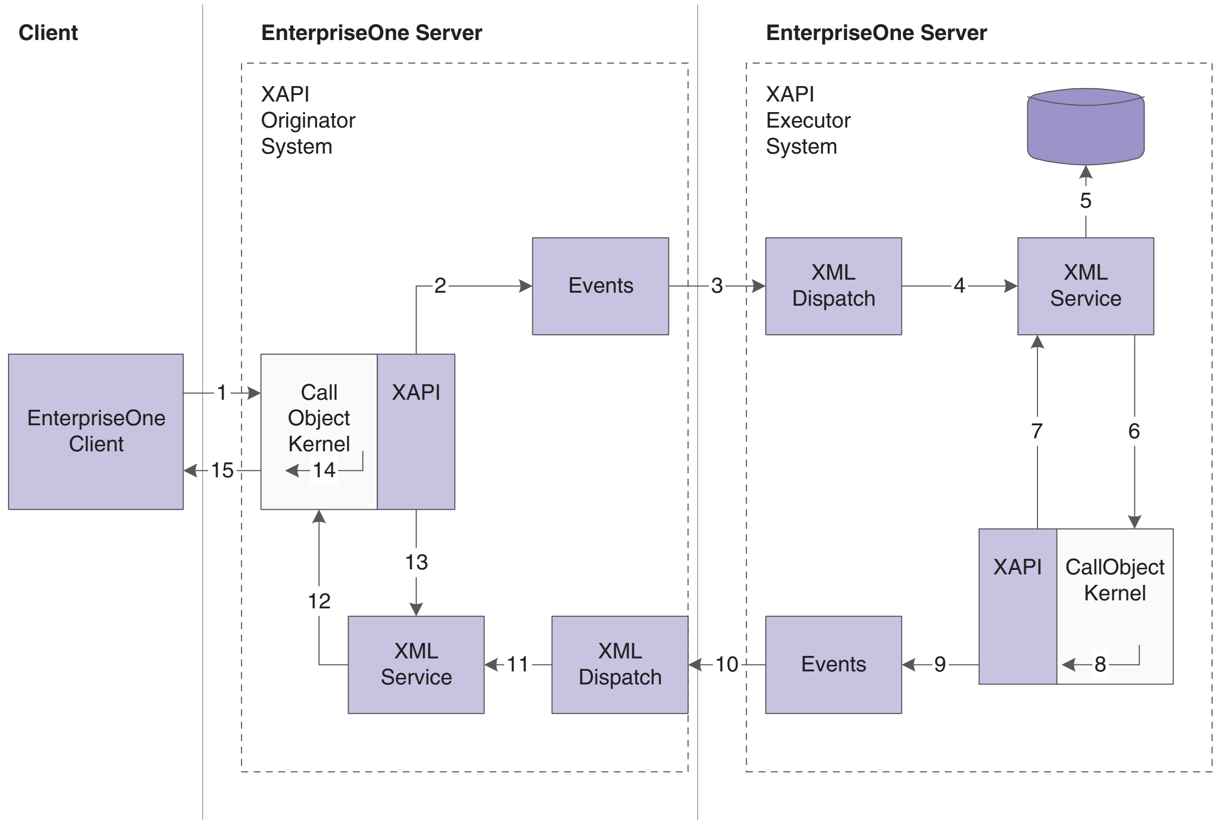
The system handles XAPI error processing for business-related errors in this manner:

- XAPI logs the business-related errors in the JD Edwards EnterpriseOne server log and these errors are delivered as part of the XAPI reply.
- XAPI APIs parse business errors from the response document.
- XAPI logs all information available about the error in the JD Edwards EnterpriseOne server log.

C.6.2 XAPI EnterpriseOne-to-EnterpriseOne Process Flow

This illustration shows a logical representation of the JD Edwards EnterpriseOne-to-EnterpriseOne XAPI process flow:

Figure C–6 JD Edwards EnterpriseOne-to-EnterpriseOne process flow



In summary:

1. For the XAPI Originator System in the illustration, a business function calls the Interoperability Event Interface within the CallObject kernel to send a request.
2. The business function uses XAPI APIs to create the XAPI request.

   XAPI adds the callback function and sends the request to the events structure.
3. The IEO kernel creates the XAPI event in XML format and sends the XML document to the EVN kernel.

   The EVN kernel ships the XML document to the XML Dispatch kernel of the second JD Edwards EnterpriseOne system. The XML document is shipped through JDENET using persistent subscription information. A routing token that contains the sender's server and port information is added. The message type for the event must be *RealTimeEvent*.
4. For the XAPI Executor System in the illustration, the XML Dispatch kernel receives the XML package and sends the event request and routing information to the XML Service kernel.
5. The XML Service kernel stores the XAPI request and creates a file handle for the XAPI request.

   The XML kernel also creates XML based routing information, stores the routing information, and creates a file handle for the routing information. The XML Service kernel uses the F907012 table to find the business function that processes the request.

6. The XML Service kernel invokes the business function (in CallObject) with the XML request handle and the routing information handle.
7. The business function uses XAPI APIs to parse and process the request. XAPI APIs load the XML request into memory.
8. The business function processes the XAPI event request.

   The business function also creates a XAPI response. The message type for the response must be *xapicallmethod.* The XAPI response is in XML format. The business function also passes the routing information handle.
9. The XAPI response originator sends the response and the routing information to the events structure.
10. The IEO kernel formats the XAPI response in XML format and sends the XML document to the EVN kernel.

    The EVN kernel uses direct routing to send the response and routing information to the XML Dispatch kernel of the first JD Edwards EnterpriseOne system (XAPI originator system). Direct routing means sending the XAPI reply to the same request-originating server.
11. For the XAPI Originator System in the illustration, the XML Dispatch kernel receives the response XML document and sends the response to the XML Service kernel.
12. The XML Service kernel stores the response document, creates a file handle, and invokes the callback business function with the file handle.
13. The business function parses the response document using XAPI APIs (XAPI response handler).

    XAPI APIs use the XML Service kernel to load the document into memory.
14. The business function uses XAPI APIs (CallObject kernel) to process the response.
15. The business function can poll for the XAPI response from the JD Edwards EnterpriseOne server.

> **Note:** You can send a request from one JD Edwards EnterpriseOne system to another JD Edwards EnterpriseOne system for processing with no return reply. If you do not want a response, use the steps through step 8 without processing the request. No response is generated.

C.6.3 XAPI Outbound Request Generation APIs

You use APIs to generate a XAPI request from the originator system. These APIs are the same as the APIs that are identified in the XAPI Outbound Events section.

- jdeXAPI_SimpleSend
- jdeXAPI_Init
- jdeXAPI_Add
- jdeXAPI_Finalize
- jdeXAPI_Free

C.6.4 XAPI Outbound Request Handling APIs

The mapped business function uses these APIs in the JD Edwards EnterpriseOne executor system to retrieve XML data from the outbound XAPI request document:

- jdeXMLRequest_GetDSCount
- jdeXMLRequest_GetDSName
- jdeXMLRequest_ParseDS
- jdeXMLRequest_DeleteXML
- jdeXMLRequest_ParseNextDSByName
- jdeXMLRequest_PrepareDSListForIterationByName

C.6.5 XAPI Outbound Request Parsing API Usage Sample Code

This code example shows the API usage for generating a outbound request from the JD Edwards EnterpriseOne originator to the JD Edwards EnterpriseOne executor:

```
#include <jde.h>

#define b0000310_c


/*****************************************************************************
*    Source File:  b0000310
*
*    Description:  Company Real Time Notification Outbound Wrapper Source File
*
*****************************************************************************/

#include <b0000310.h>
#include <B4206030.h>
#include <B4206000.h>
/**************************************************************************
*  Business Function:  CompanyRealTimeWrapper
*
*        Description:  Company Real Time Notification Outbound Wrapper
*
*         Parameters:
*           LPBHVRCOM           lpBhvrCom    Business Function Communications
*           LPVOID              lpVoid       Void Parameter - DO NOT USE!
*           LPDSD0000310A        lpDS         Parameter Data Structure Pointer
*
**************************************************************************/

 int iXMLRecordCount = 0;
 int iCurrentRecord = 0;
 NID nidDSName;
 ID idReturnValue = ER_SUCCESS;
 ID idSORecordCount = ER_ERROR; /*Return Code*/
 LPDSD4206000A lpDS;
 int lpmnJobNumber;

 MATH_NUMERIC mnBatchNumber = {0};
 unsigned long lBatchNumber = {0};
 DSD4206030A dsD4206030A = {0};
 /* CacheProcessInboundDemandRequest B4206030.c */
 DSD4206000I dsD4206000I = {0};
 /* Demand scheduling inbound DSTR */
```

```
 iXMLRecordCount = jdeXMLRequest_GetDSCount(lpDS->szXMLHandle);
 if( iXMLRecordCount > 0)
 {
  for ( iCurrentRecord = 0; iCurrentRecord < iXMLRecordCount; iCurrentRecord++)
  {
   memset((void *)(&dsD4206000I), (int)(_J('\0')), sizeof(DSD4206000I));
   memset((void *)(nidDSName), (int)(_J('\0')), sizeof(NID));
   if(jdeXMLRequest_GetDSName(lpDS->szXMLHandle,iCurrentRecord,nidDSName))
   {
    /* Retrieving data*/
    if (jdeStricmp(nidDSName, (const JCHAR *)_J("D40R0180B")) == 0)
    {
     if (jdeXMLRequest_ParseDS(lpDS->szXMLHandle,iCurrentRecord,
&dsD4206000I,sizeof(DSD4206000I)))
     {
      /* Get next number for the batch number of the inbound INVRPT
record*/
      if ( dsD4206000I.cInventoryAdvisement == _J('1'))
      {
       lBatchNumber = JDB_GetInternalNextNumber();
       LongToMathNumeric(lBatchNumber, &mnBatchNumber);
       FormatMathNumeric(dsD4206000I.szBatch,&mnBatchNumber);
      }
      /* Setup cancel flag for pending delete record */
      if ( dsD4206000I.cPendingDelete == _J('1'))
      {
       /* Flag set as 1 for any cancel demand record */
       dsD4206000I.cCancelFlag = _J('1');
      }
      else
      { /* Flag set as 9 for any non cancel demand record */
       dsD4206000I.cCancelFlag = _J('9');
      }
      /* Load parms for cache */
      //memset((void *)(&dsD4206030A), (int)(_J('\0')),
sizeof(DSD4206030A));
      I4206000_LoadParmsToCache(&dsD4206000I, &dsD4206030A);
      MathCopy(&dsD4206030A.mnJobnumberA, lpmnJobNumber);
      /* Add the DSTR to cache */
      idReturnValue = jdeCallObject( _J("CacheProcessInboundDemand
Request") ,(LPFNBHVR)NULL ,lpBhvrCom ,lpVoid ,(LPVOID)&dsD4206030A,(CALLMAP *)
NULL, (int)0,(JCHAR*)NULL ,(JCHAR*)NULL ,(int)0 );
      /* Write XML DSTR to cache fail */
      if (idReturnValue == ER_ERROR)
      {
       jdeErrorSet(lpBhvrCom, lpVoid, (ID)0, _J("032E"), (LPVOID)NULL);
      }
     }
     else
     { /* warning XML parse fail */
      jdeErrorSet(lpBhvrCom, lpVoid, (ID)0, _J("40R46"), (LPVOID) NULL);
     }
    } /* end if */
   }/* end if DS name */
  }/* end for - looping all matching XML DSTR */
  /* Ensure there is at least one record */
  idSORecordCount = ER_SUCCESS;
 }/*if( iXMLRecordCount > 0) */
 return idSORecordCount;
```

C.6.6 XAPI EnterpriseOne Originator XML Sample Code

This sample code illustrates the XAPI request document from the JD Edwards EnterpriseOne originator system to the JD Edwards EnterpriseOne executor system:

```
<?xml version="1.0" encoding="UTF-16F" ?>
<jdeRequest pwd="4f3e65076f446c5d20666f4172536518435c" role="*ALL"
type="xapicallmethod" user="PP6954083" session="" environment="DV9NIS2"
responseCreator="XAPI">
   <header>
      <eventVersion>1.0</eventVersion>
       <type>XAPIDEMO</type>
       <user>PP6954083</user>
       <role>*ALL</role>
       <application />
       <version />
       <sessionID>35087181.1050101193</sessionID>
       <environment>DV9NIS2</environment>
       <host>DEN-PP6954083B</host>
       <sequenceID>DEN-PP6954083B_3112_041120031647161</sequenceID>
       <date>04112003</date>
       <time>164716</time>
       <scope />
       <codepage>utf-8</codepage>
       <instanceInfo>
          <host>DEN-PP6954083B</host>
          <port>6025</port>
          <type>JDENET</type>
       </instanceInfo>
   </header>
   <body elementCount="3">
      <errors errorCount="4">
         <error code="041H" type="BSFN ERROR" />
         <error code="041I" type="BSFN ERROR" />
         <error code="2597" type="BSFN ERROR" />
         <error code="4136" type="BSFN ERROR" />
      </errors>
      <params type="D907001A" executionOrder="0" parameterCount="14">
         <param name="szXMLHandle">DEN-PP6954083B_||_C:\builds\B9_SP0\log\
J3E9745EE032D-00000C28-00000001-00000000000000000000FFFF0A0396A3.xml</param>
         <param name="mnAddressNumber">55617</param>
         <param name="szNameAlpha">Pradip Pandey</param>
         <param name="szNameMailing">Pradip K Pandey</param>
         <param name="szAddressLine1" />
         <param name="szAddressLine2" />
         <param name="szZipCodePostal">80237</param>
         <param name="szCity>Denver"</param>
         <param name="szState">CO</param>
         <param name="szCountry" />
         <param name="mnAmountGross">100.00</param>
         <param name="mnUnits">100.00</param>
         <param name="jdDtForGLAndVouch1">2001/01/01</param>
         <param name="cDefaultAddressLine1">9</param>
      </params>
      <params type="DXAPIROUTE" executionOrder="1" parameterCount="4">
         <param name="ClientPort">6024</param>
         <param name="ClientIP">168007331</param>
         <param name="ClientMagicNumber">1</param>
         <param name="XAPIMethodID">XAPITestResponse</param>
      </params>
   </body>
```

```
</jdeRequest>
```

C.6.7 XAPI Inbound Response Generation APIs

The JD Edwards EnterpriseOne executor system uses these APIs to generate a response:

- jdeXAPIResponse_SimpleSend
- jdeXAPIResponse_Init
- jdeXAPIResponse_Add
- jdeXAPIResponse_Finalize
- jdeXAPIResponse_Free

C.6.8 XAPI Inbound Response Parsing API Usage Sample Code

This code example shows the API usage for generating an inbound reply from the JD Edwards EnterpriseOne executor to the JD Edwards EnterpriseOne originator:

```
JDEBFRTN (ID) JDEBFWINAPI SendOrderPromiseRequest (LPBHVRCOM lpBhvrCom,
LPVOID lpVoid, LPDSD4205010 lpDS)
{
/****************************************************************
Variable declarations
****************************************************************/
  char    cPromisableLine          = ' ';
  int    nHeaderBackOrderAllowed   = ' ';
  HUSER   hUser    ;
  ID     JDEDBResult               = JDEDB_PASSED;
  BOOL    bExit                    = FALSE;
  BOOL    bB4001040Called          = FALSE;
  BOOL    bXAPIInUse               = FALSE;
  BOOL    bAtLeastOneDetail        = FALSE;

  #ifdef jdeXAPI_CALLS_ENABLED
  XAPI_CALL_ID     ulXAPICallID   = 0;
  XAPI_CALL_RETURN   eXAPICallReturn  = eEventCallSuccess;
  #endif
/****************************************************************
* Declare structures
****************************************************************/
  DSD4001040   dsD4001040     = {0};
  DSD4205020   dsD4205020     = {0};
  DSD4205040   dsD4205040     = {0}; /* Header Info */
  DSD4205050   dsD4205050     = {0}; /* Detail Info */
  DSD4205010A   dsD4205010A   = {0}; /* Query Header */
  DSD4205010B   dsD4205010B   = {0} /* Query Detail */
  DSD0100042   dsD0100042     = {0};
  LPDSD4205040H  lpDSD4205040H    = (LPDSD4205040H) NULL;
  LPDSD4205050D  lpDSD4205050D    = (LPDSD4205050D) NULL;

/****************************************************************
** Declare pointers
****************************************************************/
/****************************************************************
* Check for NULL pointers
****************************************************************/
if ((lpBhvrCom == (LPBHVRCOM) NULL) ||
 (lpVoid  == (LPVOID)  NULL) ||
```

```
 (lpDS   == (LPDSD4205010) NULL))
{
jdeErrorSet (lpBhvrCom, lpVoid, (ID) 0, "4363", (LPVOID) NULL);
return ER_ERROR;
}

/* Retrieving hUser */
JDEDBResult = JDB_InitBhvr (lpBhvrCom, &hUser, (char *)NULL,
JDEDB_COMMIT_AUTO ) ;

if ( JDEDBResult == JDEDB_FAILED )
{
jdeSetGBRError ( lpBhvrCom, lpVoid, (ID) 0, "4363" ) ;
return ER_ERROR ;
}
/*************************************************************
* Set pointers
*************************************************************/
/*************************************************************
* Main Processing
*************************************************************/
 /*----------------------------------------------------*/
 /* Setting Up ErrorCode
      */
 lpDS->cErrorCode = '0';

 /*----------------------------------------------------*/
 /* Determining if XAPI is ready to be used        */

 bXAPIInUse = FALSE;

 #ifdef jdeXAPI_CALLS_ENABLED
 if(jdeXAPI_IsCallTypeEnabled("XAPIOPOUT") &&
     jdeXAPI_IsCallTypeEnabled("XAPIOPIN") )
 {
     bXAPIInUse = TRUE;
 }
 #endif

 /*----------------------------------------------------*/
 /* Data validation and default values.           */
 /* When Display Before Accept Mode is on, validate Key */
 /* Information. Otherwise retrieve it from Header Record*/

 if((lpDS->cDisplayBeforeAcceptMode == '1')   &&
      ((MathZeroTest(&lpDS->mnOrderNumber) == 0) ||
       (IsStringBlank(lpDS->szOrderType))   ||
       (IsStringBlank(lpDS->szOrderCompany))))
 {
      bExit = TRUE;
 }
 else
 {
     MathCopy(&dsD4205040.mnOrderNumber,&lpDS->mnOrderNumber);
     strncpy(dsD4205040.szOrderType,
       lpDS->szOrderType,
        sizeof(dsD4205040.szOrderType));
     strncpy(dsD4205040.szComputerID,
       lpDS->szOrderCompany,
        sizeof(dsD4205040.szOrderCompany));
```

```
    dsD4205040.cUseCacheOrWF = lpDS->cUseCacheOrWF;
    strncpy(dsD4205040.szComputerID,
      lpDS->szComputerID,
       sizeof(dsD4205040.szComputerID));
    MathCopy(&dsD4205040.mnJobNumber,&lpDS->mnJobNumber);
    jdeCallObject( "GetSalesOrderHeaderRecord",
         NULL,
         lpBhvrCom, lpVoid,
         (LPVOID)&dsD4205040,
         (CALLMAP *) NULL,
         (int) 0,
         (char *) NULL,
         (char *) NULL,
         (int) 0 ) ;

    lpDSD4205040H = (LPDSD4205040H)jdeRemoveDataPtr(hUser,
(ulong)dsD4205040.idHeaderRecord);

    if (lpDSD4205040H == NULL)
    {
         bExit = TRUE;
    }
}

/*------------------------------------------------------*/
/* Set error if we're exiting at this point      */
if (bExit == TRUE)
{
    lpDS->cErrorCode = '1';
    /* Sales Order Header Not Found */
    strncpy(lpDS->szErrorMessageID,
       "072T",
         sizeof(lpDS->szErrorMessageID));
    if (lpDS->cSuppressError != '1')
    {
       jdeErrorSet (lpBhvrCom, lpVoid, (ID) 0, "072T", (LPVOID) NULL);
    }
}

/*------------------------------------------------------*/
/* Default Promising Flag is always 1      */
lpDS->cDefaultPromisingFlags = 1;
if (bExit == FALSE)
{

 /*------------------------------------------------------*/
 /* Call XAPIInit   */
 #ifdef jdeXAPI_CALLS_ENABLED
 if(bXAPIInUse == TRUE)
 {
    ulXAPICallID = jdeXAPI_Init( lpBhvrCom,
                  SendOrderPromiseRequest,
               "XAPIOPOUT",
               NULL,
               &eXAPICallReturn);
    if (eXAPICallReturn != eEventCallSuccess)
    {
         bExit = TRUE;
    }
 }
```

```
 #endif
 if (bExit == FALSE)
 {

    /*------------------------------------------------*/
    /* Loading Header Information           */
    I4205010_PopulateQueryHeader(lpDS,&dsD4205010A
      lpDSD4205040H,&dsD0100042,hUser,lpVoid,lpBhvrCom);
    nHeaderBackOrderAllowed = dsD4205010A.nAllowBackorders;

    /*-------------------------------------------------*/
    /* Adding Header Information     */
    #ifdef jdeXAPI_CALLS_ENABLED
    if(bXAPIInUse == TRUE)
    {
       eXAPICallReturn = jdeXAPI_Add( lpBhvrCom,
                  ulXAPICallID,
                  "SendOrderPromiseRequest",
                  "D4205010A",
                  &dsD4205010A,
                  sizeof(DSD4205010A));
       if (eXAPICallReturn != eEventCallSuccess)
       {
          bExit = TRUE;
       }
    }
    #endif
 }
}
if (bExit == FALSE)
{

 /*-----------------------------------------------------*/
 /* Loading Detail Information   */
 MathCopy(&dsD4205050.mnOrderNumber,&lpDS->mnOrderNumber);
 strncpy(dsD4205050.szOrderType,lpDS->szOrderType,
             sizeof(dsD4205050.szOrderType));
 strncpy(dsD4205050.szOrderCompany,lpDS->szOrderCompany,
             sizeof(dsD4205050.szOrderCompany));
 dsD4205050.cUseCacheOrWF = lpDS->cUseCacheOrWF;
 strncpy(dsD4205050.szComputerID,lpDS->szComputerID,
             sizeof(dsD4205050.szComputerID));
 MathCopy(&dsD4205050.mnJobNumber,&lpDS->mnJobNumber);
 if (lpDSD4205040H->cActionCode != 'A')
 {
    dsD4205050.cCheckTableAfterCache = '1';
 }
 else
 {
    dsD4205050.cCheckTableAfterCache = '0';
 }
 jdeCallObject( "GetSalesOrderDetailRecordOP",
    NULL,
    lpBhvrCom, lpVoid,
    (LPVOID)&dsD4205050,
    (CALLMAP *) NULL,
    (int) 0, (char *) NULL,
    (char *) NULL, (int) 0 ) ;

 if (dsD4205050.cRecordFound != '1')
```

```
  {
     bExit = TRUE;
     lpDS->cErrorCode = '1';
     /* Sales Order Detail Not Found */
     strncpy(lpDS->szErrorMessageID,"4162",
              sizeof(lpDS->szErrorMessageID));
     if (lpDS->cSuppressError != '1')
     {
        jdeErrorSet (lpBhvrCom, lpVoid, (ID) 0, "4162", (LPVOID) NULL);
     }
  }
  while ((dsD4205050.cRecordFound == '1') && (bExit == FALSE))
  {
     lpDSD4205050D = (LPDSD4205050D)jdeRemoveDataPtr( hUser,
(ulong)dsD4205050.idDetailRecord);
     /* Reset flags */
     cPromisableLine = '0';
     bB4001040Called = FALSE;

     /*-------------------------------------------------*/
     /* Evaluate the Record from F4211 (cDataSource = 2)*/
     /*  to find out if we should promise the line */
     /*  else find out from Order Promising Detail.*/

     if(dsD4205050.cDataSource == '1')
     {
        if (lpDSD4205050D->cOPPromiseLineYN == 'Y')
        {
           cPromisableLine = '1';
        }
     }
     else if(dsD4205050.cDataSource == '2')
     {
        MathCopy ( &dsD4001040.mnShortItemNumber,
             &lpDSD4205050D->mnShortItemNumber);
        strncpy ( dsD4001040.szBranchPlant,
           lpDSD4205050D->szBusinessUnit,
              sizeof(dsD4001040.szBranchPlant));

        jdeCallObject  ( "GetItemMasterDescUOM",
                 NULL,
                 lpBhvrCom, lpVoid,
                  (LPVOID)&dsD4001040,
                  (CALLMAP *) NULL,
                  (int) 0, (char *) NULL,
                  (char *) NULL, (int) 0 ) ;

        bB4001040Called = TRUE;

        cPromisableLine = I4205010_IsLinePromisable(lpBhvrCom,lpVoid,
                   hUser,lpDS,lpDSD4205050D, dsD4001040.cStockingType);
     }
     if (cPromisableLine == '1')
     {

       /* Set this flag if at least one promisable */
       /*   detail record exists. */
       bAtLeastOneDetail = TRUE;

       if (bB4001040Called == FALSE)
```

```
  {
    MathCopy (&dsD4001040.mnShortItemNumber,
         &lpDSD4205050D->mnShortItemNumber);
    strncpy ( dsD4001040.szBranchPlant,
         lpDSD4205050D->szBusinessUnit,
         sizeof(dsD4001040.szBranchPlant));

  jdeCallObject  ( "GetItemMasterDescUOM",
           NULL,
           lpBhvrCom, lpVoid,
           (LPVOID)&dsD4001040,
           (CALLMAP *) NULL,
           (int) 0, (char *) NULL,
           (char *) NULL, (int) 0 ) ;
  }

  I4205010_PopulateQueryDetail( lpDS,&dsD4205010B,
              lpDSD4205050D,
           &dsD4001040,
           &dsD4205010A,
              &dsD0100042,
           cPromisableLine,
              hUser,
              lpVoid,
           lpBhvrCom);

  #ifdef jdeXAPI_CALLS_ENABLED
  if(bXAPIInUse == TRUE)
  {
    eXAPICallReturn = jdeXAPI_Add( lpBhvrCom,
              ulXAPICallID,
              "SendOrderPromiseRequest",
              "D4205010B",
              &dsD4205010B,
              sizeof(DSD4205010B));
    if (eXAPICallReturn != eEventCallSuccess)
    {
         bExit = TRUE;
    }
  }
  #endif
}

/*--------------------------------------------------*/
/* Fetching the next Detail Record         */
MathCopy(&dsD4205050.mnOrderNumber,&lpDS->mnOrderNumber);
strncpy(dsD4205050.szOrderType,lpDS->szOrderType,
          sizeof(dsD4205050.szOrderType));
strncpy(dsD4205050.szOrderCompany,lpDS->szOrderCompany,
         sizeof(dsD4205050.szOrderCompany));
dsD4205050.cUseCacheOrWF = lpDS->cUseCacheOrWF;
strncpy(dsD4205050.szComputerID,lpDS->szComputerID,
         sizeof(dsD4205050.szComputerID));
MathCopy(&dsD4205050.mnJobNumber,&lpDS->mnJobNumber);
if (lpDSD4205040H->cActionCode != 'A')
{
   dsD4205050.cCheckTableAfterCache = '1';
   }
   else
   {
```

```
         dsD4205050.cCheckTableAfterCache = '0';
      }
      jdeCallObject( "GetSalesOrderDetailRecordOP",
              NULL,
              lpBhvrCom, lpVoid,
              (LPVOID)&dsD4205050,
              (CALLMAP *) NULL,
              (int) 0, (char *) NULL,
              (char *) NULL, (int) 0 ) ;
   }
   if (!bAtLeastOneDetail)
   {
      bExit = TRUE;
      lpDS->cErrorCode = '1';
      /* Sales Order Detail Not Found */
      strncpy(lpDS->szErrorMessageID,"4162",
              sizeof(lpDS->szErrorMessageID));
      if (lpDS->cSuppressError != '1')
      {
         jdeErrorSet (lpBhvrCom, lpVoid, (ID) 0, "4162", (LPVOID) NULL);
      }
   }
   if (bExit == FALSE)
   {
      #ifdef jdeXAPI_CALLS_ENABLED
      if(bXAPIInUse == TRUE)
      {
         eXAPICallReturn = jdeXAPI_Finalize( lpBhvrCom,
                 ulXAPICallID,
                 "SendOrderPromiseRequest",
                 "OrderPromiseCallback)";
         if (eXAPICallReturn != eEventCallSuccess)
         {
            bExit = TRUE;
         }
      }
      #endif
   }

   /*---------------------------------------------------*/
   /* Call B4205020 in Add Mode   */
   if((bExit == FALSE) &&
       (lpDS->cDisplayBeforeAcceptMode != '1') &&
       (lpDS->cUseCacheOrWF == '2'))
   {
       MathCopy(&dsD4205020.mnOrderNumber,&lpDS->mnOrderNumber);
       strncpy(dsD4205020.szOrderType,lpDS->szOrderType,
            sizeof(dsD4205020.szOrderType));
       strncpy(dsD4205020.szOrderCompany,lpDS->szOrderCompany,
            sizeof(dsD4205020.szOrderCompany));
      strncpy(dsD4205020.szComputerID,lpDS->szComputerID,
            sizeof(dsD4205020.szComputerID));
      MathCopy(&dsD4205020.mnJobNumber,&lpDS->mnJobNumber);

      jdeCallObject( MaintainOPWorkFile,
          NULL,
          lpBhvrCom, lpVoid,
          (LPVOID)&dsD4205020,
          (CALLMAP *) NULL,
          (int) 0, (char *) NULL,
```

```
            (char *) NULL, (int) 0 ) ;
      }
   }

/**************************************************************
Function Clean Up
**************************************************************/
  #ifdef jdeXAPI_CALLS_ENABLED
  if (eXAPICallReturn != eEventCallSuccess)
  {
   /*----------------------------------------------------*/
   /* CleanUp */
   if(bXAPIInUse == TRUE)
   {
    jdeXAPI_Free( lpBhvrCom,
        ulXAPICallID,
        "SendOrderPromiseRequest");
   }

   lpDS->cErrorCode = '1';
   /* System Error - no reasonable error messages exist. */
   strncpy(lpDS->szErrorMessageID,"018Y",
                sizeof(lpDS->szErrorMessageID));
   if (lpDS->cSuppressError != '1')
   {
       jdeErrorSet (lpBhvrCom, lpVoid, (ID) 0, "018Y", (LPVOID) NULL);
   }
  }
  #endif
  if(lpDSD4205040H != (LPDSD4205040H)NULL)
  {
     jdeFree((void *)lpDSD4205040H);
  }
  if(lpDSD4205050D != (LPDSD4205050D)NULL)
  {
     jdeFree((void *)lpDSD4205050D);
  }
 return (ER_SUCCESS);
}
```

C.6.9 XAPI Inbound Response from Originator System Sample Code

The sample code illustrates the XAPI response document from the JD Edwards EnterpriseOne executor system to the JD Edwards EnterpriseOne originator system:

```
<?xml version="1.0" encoding="UTF-16" ?>
<jdeResponse pwd="4f3e65076f446c5d20666f4172536518435c" role="*ALL"
type="realTimeEvent" user="PP6954083" session="35087181.1050101193"
environment="DV9NIS2" responseCreator="XAPI">
   <event>
      <header>
         <eventVersion>1.0</eventVersion>
         <type>XAPIDEMO</type>
         <user>PP6954083</user>
         <role>*ALL</role>
         <application>P90701XT</application>
         <version />
         <sessionID>35087181.1050101193</sessionID>
         <environment>DV9NIS2</environment>
         <host>DEN-PP6954083B</host>
```

```
         <sequenceID>DEN-PP6954083B_2864_041120031636402</sequenceID>
         <date>04112003</date>
         <time>164646</time>
         <scope />
         <codepage>utf-8</codepage>
         <instanceInfo>
            <host>DEN-PP6954083B</host>
            <port>6025</port>
            <type>JDENET</type>
         </instanceInfo>
      </header>
      <body elementCount="2">
         <detail date="04112003" name="XAPITestFunctionInitiateRequest"
time="16:39:54" type="" DSTMPL="D907001A" executionOrder="0"
parameterCount="14">
         <szXMLHandle type="String" />
         <szNameAlpha type="String">Pradip Pandey</szNameAlpha>
         <szNameMailing type="String">Pradip K Pandey</szNameMailing>
         <szAddressLine1 type="String" />
         <szAddressLine2 type="String" />
         <szZipCodePostal type="String">80237</szZipCodePostal>
         <szCity type="String">Denver</szCity>
         <szState type="String">CO</szState>
         <szCountry type="String" />
         <mnAmountGross type="Double">100.00</mnAmountGross>
         <mnUnits type="Double">100.00</mnUnits>
         <jdDtForGLAndVouch1 type="Date">2001/01/01</jdDtForGLAndVouch1>
         <cDefaultAddressLine1 type="Character" />
      </detail>
      <detail date="04112003" name="XAPITestFunctionInitiateRequest"
time="16:39:54" type="" DSTMPL="DXAPIROUTE" executionOrder="1"
parameterCount="4">
         <ClientPort type="Int">6024</ClientPort>
         <ClientIP type="Int">168007331</ClientIP>
         <ClientMagicNumber type="Int">1</ClientMagicNumber>
         <XAPIMethodID type="String">XAPITestResponse</XAPIMethodID>
      </detail>
   </body>
  </event>
</jdeResponse>
```


C.6.10 XAPI Inbound Response Handling APIs

The JD Edwards EnterpriseOne originator system uses these APIs to retrieve XML data from the inbound XAPI document and generate an inbound XAPI response:

- jdeXML_GetDSCount
- jdeXML_GetDSName
- jdeXML_ParseDS
- jdeXML_DeleteXML
- jdeXML_ParseNextDSByName
- jdeXML_PrepareDSListForIterationByName

C.6.11 XAPI Error Handling APIs

The JD Edwards EnterpriseOne executor system uses these error handling APIs:

- jdeXML_CheckSystemError

  The check system error API is for system errors. It tells the JD Edwards EnterpriseOne originator system that a system error happened in the JD Edwards EnterpriseOne executor system.

- jdeXML_GetErrorCount
- jdeXML_SetErrors

  The get error count and set errors APIs are for business errors. These two APIs, when used together, find the number of business errors and then send the errors to the BHVRCOM structure for you to resolve.

C.6.12 XAPI EnterpriseOne-to-EnterpriseOne jde.ini File Configuration

To generate XAPI events, these sections of the JD Edwards EnterpriseOne server jde.ini file must be configured:

- [JDENET_KERNEL_DEF19]
- [JDENET_KERNEL_DEF20]
- [JDENET_KERNEL_DEF22]
- [JDENET_KERNEL_DEF24]
- [JDEITDRV]
- [XAPI] - XMLDirectory setting
- [XMLLookupInfo]
- [INTEROPERABILITY] - LEVEL setting

C.6.12.1 [XAPI]

Configure this setting:

```
XMLDirectory=c:\builds\bdev\log\
```

C.6.12.2 [XMLLookupInfo]

Configure these settings:

```
XMLRequestType5=XAPICallMethod
XMLKernelMessageRange5=14251
XMLKernelHostName5=local
XMLKernelPort5=0
XMLKernelReply5=0
XMLRequestType6=realTimeEvent
XMLKernelMessageRange6=14251
XMLKernelHostName6=local
XMLKernelPort6=0
XMLKernelReply6=0
```

C.6.12.3 [INTEROPERABILITY]

Configure this setting:

```
LEVEL=DOC
```

Note: The LEVEL setting is for logging the Event XML document in the JD Edwards EnterpriseOne server for debugging purposes.

Setting the LEVEL=DOC key causes all real-time events to be written to the disk, which can cause a significant performance impact on the host server. JD Edwards suggests that you not use the LEVEL=DOC setting in a production environment or for stress testing of the QA environment.

If you are using a JD Edwards EnterpriseOne client to generate XAPI events, you must define the Client Dispatch kernel and [JDENET] sections of the client jde.ini file.

See Also:

- Configuring the jde.ini for Real-Time Events.
- XAPI Client jde.ini File Configuration.

C.7 Mapping the Business Function

This section provides an overview about mapping business functions and APIs for JD Edwards EnterpriseOne-to-EnterpriseOne XAPI events, and discusses how to enter the mapping information.

C.7.1 Understanding Business Function Mapping

When the JD Edwards EnterpriseOne executor system receives an event from the JD Edwards EnterpriseOne originator, it needs to know what business function or system API to invoke to process the request. You must map the business function or system API to the XAPI event name. You map business functions and system APIs in the F907012 table. You use the Event Request Definition program (P907012) to map business functions and APIs.

If you are mapping business functions, you enter the name of the business function. If you map APIs, you must enter the name of the API and the library where it is defined. In addition, the signature of the API must be made common, similar to the business function.

Mapping business functions enables you to point a XAPI event to a business function or system API that you wrote. You do not need to modify source code of a business function that JD Edwards delivered to you.

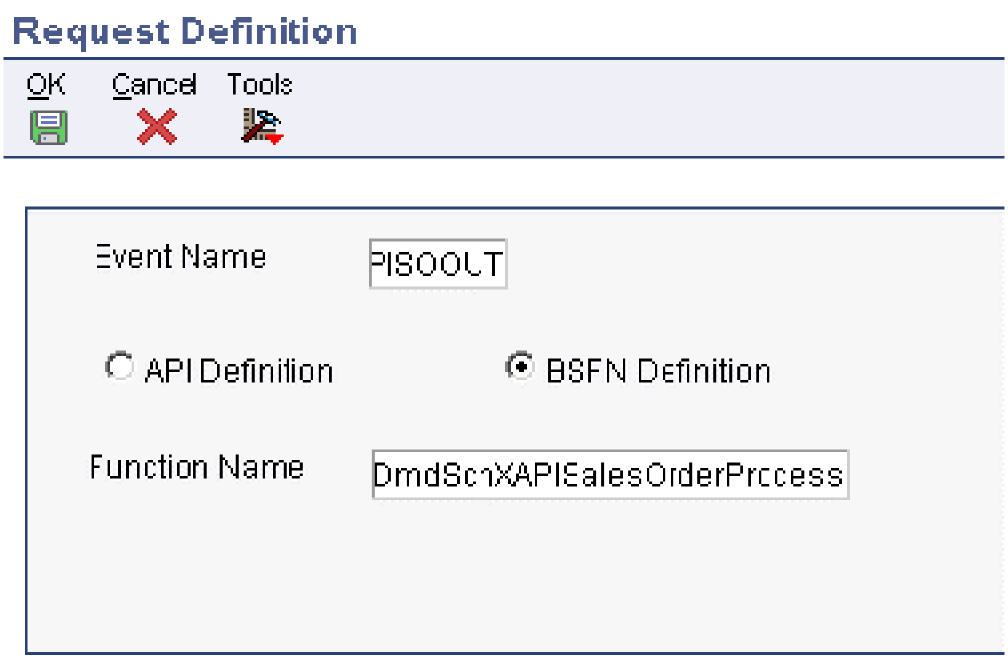
C.7.2 Forms Used to Map a Business Function or API

Form Name	FormID	Navigation	Usage
Work With Definition	W907012A	Enter P907012 in the Fast Path Command Line.	Locate and review existing mappings.
Request Definition	W907012B	On Work With Definition, click Add.	Add or change business function or API mapping for a XAPI event.

C.7.3 Mapping a business function or API

Access the Request Definition form.

Figure C–7 Request Definition form



Event Name
The name of the event (for example, JDERTSOOUT). Single events are part of other events.

BSFN Definition
An option that specifies the type of processing for an event.

API Definition
An option that specifies the type of processing for an event.

When you select the API definition option, the DLL Name field appears on the form.

Function Name
The actual name of the function. It must follow standard ANSI C naming conventions (for example, no space between words).

DLL Name
Specifies the name of the database driver file. This file is specified in the [DB SYSTEM SETTINGS] section of the enterprise server jde.ini file. The file that you specify depends upon the platform and the database. Values for specific machines and databases are:

DBDR: IBM i to DB2 for IBM i

JDBNET: IBM i to any other server DBMS

ibjdbnet.sl: HP9000 to DB2 for IBM i

libjdbnet.sl: HP9000 to Microsoft SQL Server

libora80.sl: HP9000 to Oracle (Version 8.0) UNIX

libjdbnet.so: RS6000 to DB2 for IBM i

libjdbnet.so: RS6000 to Microsoft SQL Server

libora73.so: RS6000 to Oracle (Version 7.3) UNIX

libora80.so: RS6000 to Oracle (Version 8.0) UNIX

jdbodbc.dll: Intel to IBM i

jdboci32.dll: Intel to Oracle (Version 7.2) NT

jdboci73.dll: Intel to Oracle (Version 7.3) NT

jdboci80.dll: Intel to Oracle (Version 8.0) NT

dbodbc.dll: Intel to SQL Server NT

jdbnet.dll: Digital Alpha to IBM i

jdboci32.dll: Digital Alpha to Oracle (Version 7.2) NT

jdboci73.dll: Digital Alpha to Oracle (Version 7.3) NT

jdboci80.dll: Digital Alpha to Oracle (Version 8.0) NT

dbodbc.dll: Digital Alpha to SQL Server NT

D

Using Classic Z Events

This appendix contains the following topics:

Note: This chapter is applicable only if you use classic events delivery. Classic event delivery is available when you use JD Edwards EnterpriseOne Tools 8.93 or earlier releases, or if you use JD Edwards EnterpriseOne Tools 8.94 with JD Edwards EnterpriseOne Applications 8.10.

Refer to the Guaranteed Events chapters if you use JD Edwards EnterpriseOne Tools 8.94 with JD Edwards EnterpriseOne Applications 8.11 or JD Edwards EnterpriseOne Tools 8.95 and later tools releases with JD Edwards EnterpriseOne Applications 8.10 and later Applications releases.

D.1 Understanding Z Events - Classic

A Z event is near real-time notification that an interoperability transaction has occurred. To generate Z events, JD Edwards EnterpriseOne uses the Z event generator and the existing interface table infrastructure. You can use the existing JD Edwards EnterpriseOne interface tables, or you can build customized interface tables as long as the tables are created using JD Edwards EnterpriseOne standards.

Z event XML documents use the JD Edwards EnterpriseOne XML Response format. An example of the Z event XML document can be found in Appendix E, XML Format Examples (Events). Different events can have different table names and column names.

D.1.1 Prerequisites

Before you complete the tasks in this section:

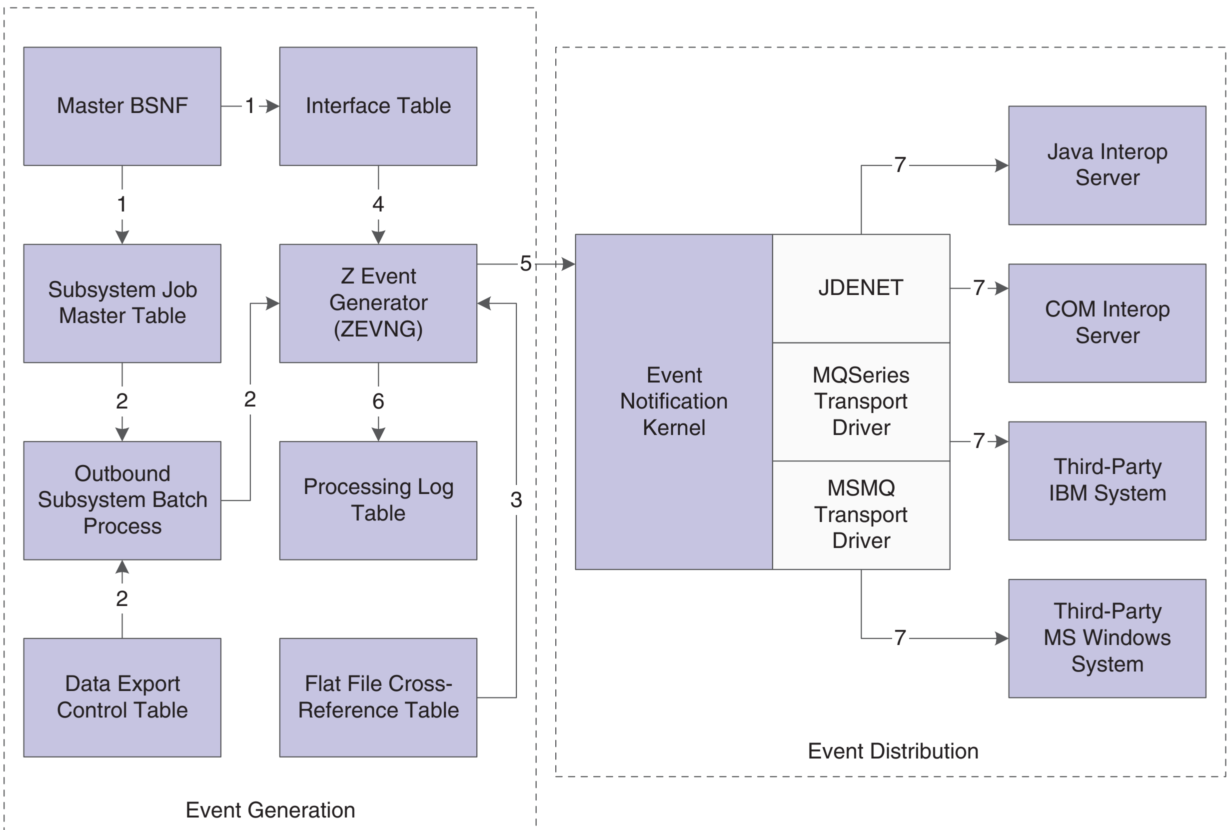
- You must enable security for the JD Edwards EnterpriseOne server.

- You must have a valid security record for the default user under the [SECURITY] section of the JD Edwards EnterpriseOne server jde.ini file (that is, the user must be a valid JD Edwards EnterpriseOne user).

D.2 Z Event Process Flow

This diagram depicts a logical representation of the processes and data for Z event generation:

Figure D–1 Z event process flow



In summary:

1. When a JD Edwards EnterpriseOne transaction occurs, the master business function writes the transaction information in the appropriate interface table and sends an update record to the F986113 table.
2. A batch process monitors the F986113 table.

   When the batch process finds a W status in the F986113 table, it notifies the Z event generator. The batch process accesses the F0047 table to determine which Z-event generator to call.
3. The F47002 table provides a cross-reference between the transaction and the interface table where the record is stored.

   This information is used by the Z-event generator.

4. The Z-event generator retrieves the transaction information from the interface table and converts the transaction information into an XML document using a JD Edwards EnterpriseOne DTD.
5. The Z-event generator sends the event (in the form of an XML document) to the event notification kernel for distribution.
6. After an event is successfully generated, the system updates the F0046 table.

   A UBE purges information in the interface table based on information in the Processing Log table.
7. The event notification kernel sends the XML document to all subscribers.

> **Note:** If you use IBM WebSphere MQ or MSMQ transports, the transport driver writes system and function errors to the JDE error log. The driver writes error messages and adds the error codes, if available.

D.3 Z Event Sequencing

When you define your Z events, you indicate whether the event is reliable or volatile. If you define the event as volatile, the system automatically provides event sequencing to guarantee that events are delivered in the correct order. Volatile events are stamped using features of JD Edwards EnterpriseOne Next Numbers.

For sequencing of Z events, ZEVG, the Z event generator, retrieves the next number from the Z event sequencing bucket and sends the number to the EVN kernel for sequencing purposes. It is important to note that JD Edwards only guarantees the sequence for the particular type of event generator. This is due to the inherent delays that are involved in the Z event processing; an event that occurred earlier can get a later sequence number.

Event sequencing does affect performance. You can clear events sequencing. You can also define a timeout value to tell the system to stop looking for a missed event when events are out of sequence. The flag and timeout settings are in the [INTEROPERABILITY] section of the jde.ini file.

D.4 Vendor-Specific Outbound Functions

The purpose of the vendor-specific outbound function is to pass the key fields for a record in the outbound interface tables to a third-party system. With these keys, you can process information from the database record into your third-party system. The generic Outbound Subsystem batch process calls the function.

Each vendor-specific function is specific to the transaction being processed. You must decide how the function actually uses the database record information. Although the functions are written to your specifications, and most likely are written outside of JD Edwards EnterpriseOne, these functions must use the required JD Edwards EnterpriseOne defined data structure as illustrated in this table:

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

Data Item	Required	I/O	Description
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

D.5 Working With Z Events

This section provides an overview of Z event processing and discusses how to set up a data export control record.

D.5.1 Understanding Z Event Processing

To use Z events to retrieve information from JD Edwards EnterpriseOne, perform these tasks:

- Enable Z event processing.
- Update Flat File Cross-Reference table.
- Update the Processing Log table.
- Verify that the subsystem job is running.
- Purge data from the interface table.
- Configure the jde.ini file for Z events.
- Set up data export controls.

D.5.2 Enabling Z Event Processing

You can enable or disable master business functions to write transaction information into interface tables and the F986113 table. when a transaction occurs. All outbound master business functions that have the ability to create interoperability transactions have processing options that control how the transaction is written. On the Processing Options Interop tab, the first processing option is the transaction type for the interoperability transaction. If you leave this processing option blank, the system does not perform outbound interoperability processing. The second processing option controls whether the *before* image is written for a change transaction. If this processing option is set to *1*, the system writes before and after images of the transaction to the interface table. If this processing option is not set, then the system writes only an *after* image to the interface table.

See Interoperability Interface Table Information.

D.5.3 Updating Flat File Cross-Reference Table

When you enable Z events, you also update the F47002 table. The transaction type that you entered in the processing option maps to the F47002 table to determine which interface tables to use to retrieve the information. You use the Flat File Cross-Reference program (P47002) to update the F47002 table.

D.5.4 Updating the Processing Log Table

The Z event generator uses the F0046 table. The F0046 table contains the keys to the interoperability transaction along with a successfully processed column. The sequence number, transaction type, order type, function name, and function library are obtained from the F0047 table. A vendor-specific record is sequentially created in the F0046 table for every transaction that is processed by the Interoperability Generic Outbound Subsystem (R00460) UBE or the Interoperability Generic Outbound Scheduler UBE (R00461).

For example, if three vendors have subscribed to a transaction using the F0047 table, the system creates three records in the F0046 table, (one record 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 can use the Processing Log (P0046) program to determine whether a vendor-specific object correctly processed the interoperability transaction.

A purging UBE that purges the interfaces tables runs based on information in the processing log table.

Data in the Processing Log table cannot be changed.

D.5.5 Verifying that the Subsystem Job is Running

When the application master business function adds a record to the F986113 table, the system starts a subsystem job. Subsystem jobs are continuous jobs that process records from the F986113 table. You should verify that the subsystem job is running.

Note: After the records are processed, instead of ending the job, subsystem jobs look for new data in the data queue. Subsystem jobs run until you terminate them.

You can schedule subsystem jobs.

See "Understanding JD Edwards EnterpriseOne Subsystems" in the *JD Edwards EnterpriseOne Tools System Administration Guide*.

See "Understanding the Job Scheduler" in the *JD Edwards EnterpriseOne Tools System Administration Guide*.

D.5.6 Purging Data from the Interface Table

After you receive the Z event, you should purge the data from the interface table. You can enter a purge UBE in the F0046 table to purge the interface table.

See Purging Interface Table Information.

See Interoperability Interface Table Information.

D.5.7 Configuring the jde.ini File for Z Events

To generate Z events, you must configure these sections of the JD Edwards EnterpriseOne server jde.ini file:

- [JDENET_KERNEL_DEF19]
- [JDEITDRV]
- [JDENET]
- [INTEROPERABILITY]

The settings for the EVN kernel, [JDEITDRV], and [JDENET] are defined in the jde.ini File Configurations for Events section of this guide. You must configure settings for [INTEROPERABILITY].

D.5.7.1 [INTEROPERABILITY]

Configure these settings:

```
SequenceTimeOut=XX
XMLElementSkipNullOrZero=X
```

The SequenceTimeOut setting is for sequencing of volatile events. The value is in seconds.

Null strings and zeros are trimmed off Z events. You can clear this feature by entering a value of 0 (zero) for the XMLElementSkipNullOrZero setting.

D.6 Setting Up Data Export Controls

This section provides an overview of setting up data export controls and discusses how to set up the record.

D.6.1 Understanding Data Export Controls Records

The generation of outbound data is controlled through the F0047 table. You use the Data Export Controls program (P0047)to update the F0047 table. For each transaction type and order type, you must designate the Z event generator that processes the outbound data. To send a given transaction type to more than one third-party application, you associate the transaction type with each of the individual destinations by making separate entries for each destination in the F0047 table. JD Edwards suggests that you 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 that you can retrieve the transaction.

D.6.2 Forms Used to Add a Data Export Controls Record

Form Name	FormID	Navigation	Usage
Work with Data Export Controls	W0047A	From a application that supports event generation, open the Data Export Controls Program An alternative way to access the Data Export Controls Program is to enter P0047 in the Fast Path command line	View existing data export control records.
Data Export Control Revisions	W0047C	On Work with Data Export Controls, click Add.	Add a new data export control record.

D.6.3 Adding a Data Export Control Record

Access the Data Export Control Revisions form.

To set up Data Export Controls:

1. Complete these fields:
   - Transaction
   - Order Type
2. For each detail row, enter one of these, depending on your platform:
   - Function Name

     Windows NT: _CallOnUpdate@36

     UNIX: CallOnUpdate

     IBM i: CallOnUpdate
   - Function Library

     Windows NT: EnterpriseOne Bin32 Path\zevg.dll

     UNIX(HP): EnterpriseOne Bin32 Path\libzevg.sl

     UNIX(AIX, SUN): EnterpriseOne Bin32 Path\libzevg.so

     IBM i: EnterpriseOne Bin32 Path\ZEVG
   - Enter **1** in the Execute For Add column to generate an event for an add/insert.

     Complete the same process as appropriate for update, delete, and inquiry.
   - 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.

     The system automatically increments the Sequence field for each line.

E

Events Self-Diagnostic Utility Tool

This appendix contains the following topics:

E.1 Understanding the Events Self-Diagnostic Utility Tool

The Events Self-Diagnostic Utility Tool supports Z events and real-time events. Normally, your system administrator runs the Self-Diagnostic Utility Tool to verify that your events infrastructure features are functional. The Self-Diagnostic Utility Tool can be used on these platforms:

- Windows 2000 and NT
- IBM i
- HP
- Sun
- AIX

The Events Self-Diagnostic Utility Tool analyzes the infrastructure of an event and reports configuration, kernel, and network problems that are detected as the event is processed through your system. You can use the tool to perform a comprehensive analysis, or you can configure the tool to perform an analysis that is specific for your needs. The Events Self-Diagnostic Tool uses the XML comparator to compare XML documents to detect the presence of any data corruption in event information. The tool also suggests actions that you can take to resolve problems. You can run this tool on either a server or a client or both.

E.2 Events Self-Diagnostic Utility Tool Process

After an event is generated at the call object API on the server or the application API on the client, problems that cause the event to fail can occur. This list identifies problems that might occur:

- The jde.ini file has a configuration error.
- The ZEVG library is unavailable or the IEO or EVN kernel process is down.

- Subscribers and supported events have not loaded successfully.
- One or more of the kernels involved in the event delivery is corrupting the event information.
- The network link between any or all of the components involved in this infrastructure is permanently down.

When the Events Self-Diagnostic Tool detects a problem, the tool sends messages to you explaining the problem and suggesting resolutions and also logs the error in the appropriate log files. The message that is sent to you indicates which log files you should review. This list provides some examples of how the Events Self-Diagnostic Tool detects problems:

- Performs an in-depth interoperability-oriented analysis of the jde.ini file.
- Reads the F90701 table to determine whether the event is defined.
- Reads the F90702 to determine whether the persistent subscription/unsubscription request, which is sent to the EVN kernel by the tool, is successful.
- Reads the Object Configuration Manager to find the location of the IEO kernel.

  In this process, the tool ensures there is only one active entry for the RTE object.
- Checks inter-connectivity within events infrastructure by sending self-diagnostic connectivity message calls.
- Generates self-diagnostic events to test different services that are offered by the infrastructure and to verify event information against possible data corruption.

Note: This list is general and not all-inclusive.

E.3 Events Self-Diagnostic Utility Tool Components

The Events Self-Diagnostic Utility Tool consists of three components:

- Event generator
- Event receiver
- XML comparator

E.3.1 Event Generator

The Events Self-Diagnostic Utility Tool starts with an event generator process. During startup, the event generator performs basic background analysis of the events infrastructure, which include:

- Verification of interoperability specific sections of the jde.ini file.
- Verification of real-time events definition.
- Inter-component connectivity check within the events infrastructure.

If startup is successful, the event generator tests different features offered by the events infrastructure. These features include generating and testing different types of events, listing the valid events, checking the event template, and testing subscription information. You can run one or more of these tests by using one of these methods:

- Running the test against an existing configuration file that you previously set up.
- Running the test against a new configuration file, which you will set up.

- Selecting options and executing the test from the command line menu of the tool.

After successful generation of a self-diagnostic event, the event is passed through the event infrastructure system. To test the accuracy of the event information that is being conveyed through the system, the event generator attaches an additional packet, in the form of XML stream, to the event. The diagnostic XML packet contains information about the event. At each stage of communication, each kernel (or component) verifies the event information by comparing standard message packets with the self-diagnostic XML packet. The kernel (or component) logs the result of this comparison at each point of comparison in respective log files. The accuracy of the information in template requests is tested the same way.

E.3.2 Event Receiver

The event receiver acts as a NULL transport driver that subscribes itself for self-diagnostic events during EVN kernel startup. The event receiver compares and verifies the XML documents contained in the received self-diagnostic events. The event receiver logs the result of this comparison in the EVN kernel log file.

E.3.3 XML Comparator

The event generator uses the XML comparator tool to test the accuracy of event information or of an event template request that is being passed through the system. The XML comparator compares any two given XML documents for equivalency, similarity, or both. To perform the comparison, the XML comparator requires three XML documents. Two of the documents are the actual XML documents to be compared. The third document is an exclusion XML document that contains nodes that are to be ignored during the comparison of the two given XML documents.

E.4 Customizing the Tool

When you select the Customize Tool option from the Interface menu, the tool prompts you to provide a new file name or to use an existing configuration file (one that you previously created using this tool). The tests (actions) and options for each test are the same tests that are discussed previously.

To use an existing configuration file (an XML file that you previously created), type the filename at the prompt and press Enter or Return. This action starts the diagnosis against your previously built configuration file.

To create a new configuration file, type a filename using XML as the extension, and then press Enter or Return. The tool offers the same tests that are on the Command Line Execution menu. You can select one or more tests by using a comma to separate the test numbers.

E.5 Executing the Events Self-Diagnostic Tool

This section provides overviews of each of the self-diagnostic tests that you can run when you execute the Events Self-Diagnostic Tool

E.5.1 Executing the Event Self-Diagnostic Tool

To use the Event Self-Diagnostic Tool, you must have a valid JD Edwards EnterpriseOne user ID, password, environment, and role. If you are using the tool from a JD Edwards EnterpriseOne server and you do not supply this information as parameters, the user name, password, environment, and role information is read from the security section of the server jde.ini file. If you are using a client, you must enter a valid JD Edwards EnterpriseOne user name, password, environment, and role. If you do not enter this information, the tool stops. If you are generating events from a client, you must also have a valid OCM mapping for RTE or Z events to a valid server. Before you run the Events Self-Diagnostic Tool:

- Ensure PORTTEST runs successfully on your system.
- Ensure that one instance of the IEO and EVN kernel is running.
- Ensure self-diagnostic events are defined in the F90701) table.

E.5.2 Start the Tool

To start the Events Self-Diagnostic Tool on the JD Edwards EnterpriseOne Server, double-click the executable file at this location: `$system\bin32\sdtool.exe`

Or you can pass parameters, for example: `$system\bin32\sdtool.exe username password environment role`

To start the tool from the client side, you must include these parameters: `$system\bin32\sdtool.exe username password environment role`

Note: $system refers to the path where the application is installed on your system.

The Events Self-Diagnostic Tool accesses the Security section of the jde.ini file for a valid user name, password, environment, and role. Upon startup, the tool analyzes the jde.ini file, verifies that events are defined, and checks the inter-component connectivity within the events infrastructure. As the tool analyzes each of these areas, it provides you with feedback about what is being analyzed and whether the analysis was successful.

If the tool detects a problem in any one of the startup areas, the tool terminates the diagnosis and sends you a message that explains the problem encountered and suggesting actions for resolving the problem.

After successful startup, you have a choice of creating and using a customized configuration file or using the command line of the tool to run the diagnosis. The Customized Tool option enables you to build and save a diagnostic test to a file so that you can run that test as often as needed without having to reenter information into the tool. When you use the Command Line Execution option, you must enter the test information when the tool prompts you. When you run tests from the command line, the Interface menu always follows the results statements so that you can run another test or exit the tool.

Whether you select Option 1, Customize(d) Tool, or Option 2, Command Line Execution, the tests that the tool performs are the same.

You can select one or more tests by typing the number that is associated with the test at the prompt and then pressing Enter or Return. For multiple tests, separate the number of the test with a comma (,). Some of the tests provide further options. At the prompt, you enter one or more options, using a comma to separate multiple options. The tool performs the test and provides feedback to you indicating success or failure. If the test failed, the tool provides feedback that tells you that the test failed and identifies the logs you should review for more information.

E.5.3 Generate/Test Real-Time Event

When you select Action 1, Generate/Test Real Time Event(s), the tool displays the real-time event types from which you select one or more real-time event types to test.

The tool generates the real-time event you requested and attaches a self-diagnostic XML document to the event. The event contents are verified for any data corruption against the attached XML document at each kernel in the events infrastructure and event receiver transport driver. You receive a message indicating whether the event was successfully generated. You also receive this feedback message: `Please see log files corresponding to IEO and EVN for any present event data corruption`. This message tells you to look at the log files to determine whether an XML document mismatch occurred. The tool also provides a final message that indicates that the tool has completed the analysis for that action, and then it returns you to the tool interface menu.

E.5.4 Generate/Test Z Event

When you select Action 2, Generate/Test Z Event, the tool displays another set of options. You can test a simulated Z event and you can test a Z event that uses the actual interface tables (Z tables). If you test a simulated Z event, the tool generates a simulated Z event and attaches a self-diagnostic XML document to the event. The event contents are verified for any data corruption against the attached XML document at the EVN kernel and event receiver transport driver. You receive a message indicating whether the event was successfully generated. You also receive this feedback message: `Please see log files corresponding to EVN for any present data corruption`. This message tells you to look at the EVN log file to determine whether an XML document mismatch occurred. The tool also provides a final message that indicates that the self-diagnostic tool has completed the analysis for the action, and then it returns you to the tool interface menu.

If you generate an actual Z event, you must have a valid UBE and you must set up the appropriate interface tables. The tool asks you for your user name, batch number, transaction number, line number, transaction type, document type, and sequence number. The tool uses the live interface tables (Z tables) for this test. When you request an actual Z event, the tool generates the Z event but does not include a self-diagnostic XML document. The EVN kernel returns a message that indicates whether the event was successful. Because no self-diagnostic XML document exists, the tool cannot diagnose data corruption.

E.5.5 Test All Types of Events

When you select Action 3 (Test all types of events) from the Execution menu, the tool tests all of the real-time events (single, aggregate, and composite) and both Z events (simulated and actual). Action 3 is the same as Action 1 (all three options) and Action 2 (both options) combined. For the real-time events and the simulated Z event, the tool generates the event and attaches a self-diagnostic XML document to the event so that any data corruption can be detected. If you select this action, you must have a valid UBE and you must set up appropriate interface tables. If you run this test but do not have actual Z event data, you can abort that portion of the test by entering Exit when the tool asks for the Z event information.

The tool sends the event information to the IEO and EVN kernels, and the kernels return messages indicating whether each event was successful.

E.5.6 Get Event List

When you select Action 4, Get Event List (List of events supported) from the Execution menu, the tool immediately runs the test.

The tool sends a getEventList request to the EVN kernel. The EVN kernel responds with the list of event names that you have defined. To validate that the list is complete, the tool checks the list for the presence of self-diagnostic event names. The tool provides a list of the events to you along with a message indicating whether the test was successful.

E.5.7 Get Event Template

When you select Action 5, Get Event Template, the tool displays the real-time event types that have a template associated with them.

The tool generates the template request and attaches a self-diagnostic XML document to the request. The template request is verified for any data corruption against the attached XML document at each kernel in the events infrastructure and event generator. The tool provides feedback that the template request was successful and that the template data was validated against the XML packet. If the test fails, the tool provides a message that indicates the reason for the failure.

E.5.8 Subscription Services

When you select Action 6, Subscription Services, the tool displays a set of options for the type of subscription service to be tested.

When you select the Persistent Subscribe option, the tool sends a persistent subscription request for a registered self-diagnostic event to the EVN kernel. The tool verifies the list of subscribers that are maintained in a file or from the database table (depending on how your system is configured), and then sends you a message indicating whether the test was successful.

When you chose the Persistent Unsubscribe option, the tool sends a persistent unsubscription request for a registered self-diagnostic event to the EVN kernel. The tool verifies that the subscription is no longer in the file or database table (depending on how your system is configured), and then sends you a message indicating whether the test was successful.

When you select the Non-Persistent Subscribe option, the tool sends a non-persistent subscription request for a registered self-diagnostic event to the EVN kernel. The tool verifies the list of subscribers that is kept by the EVN kernel, and then sends you a message indicating whether the test was successful.

When you select the Non-Persistent Unsubscribe option, the tool sends a non-persistent unsubscription request for a registered self-diagnostic event to the EVN kernel. The tool verifies the subscription is no longer in EVN, and then sends you a message indicating whether the test was successful.

E.5.9 Comprehensive System Analysis

When you select Action 7, Comprehensive System Analysis, the tool performs all of the tests and provides messages to you indicating whether each test was successful.

F

Interoperability Interface Table Information

This appendix contains the following topics:

- Section F.1, "Interoperability Interface Table Information"

F.1 Interoperability Interface Table Information

This section provides a table that lists applications that have interoperability features.

Program	Interface Table (Z table)	Input Subsystem Batch Process	Input Processor Batch Process	Extraction Batch Process	Revisions Program	Purge Batch Process	Program with POs
Financials	N/A	N/A	N/A	N/A	N/A	N/A	N/A
Address Book	F0101Z2	R01010Z - ZJDE0002	R01010Z - ZJDE0001	N/A	P0101Z1	R0101Z1P	P0100041
Customer Master	F03012Z1	R03010Z - ZJDE0002	R03010Z - ZJDE0001	N/A	P0301Z1	R0101Z1P	P0100042
Supplier Master	F0401Z1	R04010Z - ZJDE0002	R04010Z - ZJDE0001	N/A	P0401Z1	R0101Z1P	P0100043
A/R Invoice	F03B11Z1, 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	N/A	P0413Z1	R0413Z1	P0413M
Journal Entry	F0911Z1, F0911Z1T	R09110Z - ZJDE0005	R09110Z - ZJDE0002	N/A	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	N/A	N/A	N/A	N/A	N/A	N/A	N/A
Payroll Time Entry	F06116Z1	R05116Z1I	R05116Z1I - ZJDE0001	N/A	P05116Z1	R05116Z1P	N/A

Program	Interface Table (Z table)	Input Subsystem Batch Process	Input Processor Batch Process	Extraction Batch Process	Revisions Program	Purge Batch Process	Program with POs
Distribution	N/A	N/A	N/A	N/A	N/A	N/A	N/A
Purchase Order	F4301Z1, F4311Z1	R4311Z1I - XJDE0002	R4311Z1I - XJDE0001	N/A	P4311Z1	R4301Z1P	P4310
Outbound Purchase Receipts	F43121Z1	N/A	N/A	N/A	P43121Z1	R43121Z1P	P4312
Receipt Routing	F43092Z1	R43092Z1I - XJDE0002	R43092Z1I - ZJDE0001	N/A	P43092Z1	R43092Z1P	P43250
Outbound Sales Order	F4201Z1, F4211Z1, F49211Z1	N/A	N/A	N/A	P4211Z1	R4211Z1P	P4210
Outbound Shipment Confirmation	F4201Z1, F4211Z1, F49211Z1	N/A	N/A	N/A	P4211Z1	R4211Z1P	P4205
Logistics	N/A	N/A	N/A	N/A	N/A	N/A	N/A
Cycle Counts	F4141Z1	R4141Z1I	R4141Z1I - ZJDE0001	N/A	P4141Z1	R4141Z1P	N/A
Item Master	F4101Z1, F4101Z1A	R4101Z1I	R4101Z1I - ZJDE0001	N/A	P4101Z1	N/A	P4101
Item Cost	F4105Z1	N/A	R4105Z1I - XJDE0001	N/A	P4105Z1	R4105Z1P	P4105
Warehouse Confirmations (Suggestions)	F4611Z1	R4611Z1I	R4611Z1I - ZJDE0001	N/A	P4611Z1	R4611Z1P	N/A
Manufacturin g	N/A	N/A	N/A	N/A	N/A	N/A	N/A
Work Order Header	F4801Z1	Use Work Order Completions	Use Work Order Completions	R4101Z1O	P4801Z1	R4801Z1P	P48013
Work Order Parts List	F3111Z1	Use Planning Messages	Use Planning Messages	N/A	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	N/A	P31122Z1	R31122Z1	P311221
Work Order Inventory Issues	F3111Z1	R31113Z1I - ZJDE0002	R31113Z1I - ZJDE0001	N/A	P3111Z1	R3111Z1P	N/A
Work Order Completions	F4801Z1	R31114Z1I - XJDE0002	R31114Z1I - XJDE0001	N/A	P4801Z1	R4801Z1P	N/A
Super Backflush	F3112Z1	R31123Z1I	R31123Z1I - ZJDE0001	N/A	P3112Z1	R3112Z1P	N/A
Bill of Material	F3002Z1	R3002Z1I - ZJDE0002	R3002Z1I - ZJDE0001	N/A	P3002Z1	R3002Z1P	P3002

Program	Interface Table (Z table)	Input Subsystem Batch Process	Input Processor Batch Process	Extraction Batch Process	Revisions Program	Purge Batch Process	Program with POs
Routing Master	F3003Z1	R3003Z1I - ZJDE0002	R3003Z1I - ZJDE0001	N/A	P3003Z1	R3003Z1P	P3003
Work Center Master	F30006Z1	R30006Z1I - ZJDE0002	R30006Z1I - ZJDE0001	N/A	P30006Z1	R30006Z1P	P3006
Work Day Calendar	F0007Z1	R0007Z1I - XJDE0002	R0007Z1I - XJDE0001	N/A	P0007Z1	R0007Z1P	P00071
Planning Messages	F3411Z1	R3411Z1I - ZJDE0002	R3411Z1I - ZJDE0001	N/A	P3411Z1	R3411Z1P	N/A
Detail Forecast	F3460Z1	R3460Z1I - XJDE0002	R3460Z1I - XJDE0001	N/A	P3460Z1	R3460Z1P	P3460, R3465, R34650 (Each done individually)
Kanban Transactions	F30161Z1	R30161Z1I - XJDE0002	R30161Z1I - XJDE0001	N/A	P30161Z1	R30161Z1P	N/A

G

XML Format Examples (All Parameters)

This appendix contains the following topics:

- Section G.1, "Inbound Sales Order XML Format (All Parameters)"
- Section G.2, "Outbound XML Request and Response Format (All Parameters)"

G.1 Inbound Sales Order XML Format (All Parameters)

This section provides example code for an inbound sales order. This sample code shows the XML format with all of the parameters.

```
"<?xml version='1.0'?>
<jdeRequest type='callmethod' user='userid' pwd='password'
environment='environment' role='*ALL'>
<callMethod name='GetLocalComputerId' app='NetCommerce' runOnError='no'>
<params>
<param name='szMachineKey'id='2'></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>         ( 1 = Full)
 <param name='szCMComputerID' idref='c2'></param>
 <param name='cCMErrorConditions'>value</param>  (1=Warnings, 2=Errors)
 <param name='cCMUpdateWriteToWF'>value</param>  (1=wf,2=cache)
 <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>         (If blank def Proc Opt)
 <param name='szBusinessUnit'>value</param>      (If blank def Proc Opt)
 <param name='szOriginalOrderCo'>value</param>   (used copy/blanket function)
 <param name='szOriginalOrderNo'>value</param>   (used copy/blanket function)
 <param name='szOriginalOrderType'>value</param> (used copy/blanket function)
 <param name='mnAddressNumber'>value</param>     (Required if ship to = 0)
 <param name='mnShipToNo'>value</param>          (Required if sold to = 0)
 <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>  (Required, blank = parms )
<param name='cWKProcMode'>blank</param>      (blank = reg order)
<param name='mnWKSuppressProcess'>0</param>  (0 = def, 2=P/O)
<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>
  <param name='mnTransactionID'>value</param>
  </params>
 </callMethod>
</onError>
</callMethod>
<callMethod name='F4211FSEditLine'app='NetCommerce' runOnError='yes'> (each line)
 <params>
  <param name='mnCMJobNo' idref='j1'></param>
  <param name='cCMLineAction'>value</param>
  <param name='cCMProcessEdits'>value</param>
  <param name='cCMWriteToWFFlag'>value</param>
  <param name='cCMRecdWrittenToWF'>value</param>
  <param name='szCMComputerID' idref='c2'></param>
  <param name='cCMErrorConditions'>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='mnWKTotalCost'>value</param>
<param name='mnWKForeignTotalCost'>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='mnWKSuppressProcess'>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='2'></param>
  <param name='cCMErrorCondition'>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>  (Skip Proc Opt, 1="Yes")
  <param name='mnWKRelatedOrderProcess'>value</param>
  <param name='cCMUseWorkFiles'>value</param>(Req,Work File="1", Cache ="2")
  <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='2'></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='messsage name'
failureDestination='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='2'></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='mnProcessID'>value</param>
    <param name='mnTransactionID'>value</param>
  </params>
 </callMethod>
</onError>
</jdeRequest>
```

G.2 Outbound XML Request and Response Format (All Parameters)

This section provides example request and response code that illustrate the outbound XML Format with all of the parameters.

G.2.1 Request

This format returns all columns for the F0101Z2 table:

```
<?xml version='1.0' ?>
<jdeRequest type='trans' user='user' pwd='password' environment='environment'
role='*ALL' session='' sessionidle='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>
```

G.2.2 Response

This sample code shows the response for the request:

```
<?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>
<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='GlBankAccount'></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>
```

H

Minimum Required Values Sample Code

This appendix contains the following topics:

- Section H.1, "Sales Order Minimum Required Values"

H.1 Sales Order Minimum Required Values

This sales order entry example shows the minimum required parameters. JD Edwards EnterpriseOne recommends that you start with the minimum required values and test them to ensure your system is working. After you are confident the minimum required values are working properly, you can add other values.

```
<?xml version="1.0" encoding="utf-8" ?>
  <jdeRequest type="callmethod" user="JDE" pwd="JDE" role="*ALL"
environment="PRD733">
   <callMethod name="GetLocalComputerId" app="NetComm" runOnError="no">
   <params>
     <param name="szMachineKey" id="" />
   </params>
   <onError abort="yes"/>
   </callMethod>
   <callMethod name="F4211FSBeginDoc" app="NetComm" runOnError="no">
   <params>
     <param name="szCMComputerID" idref="2"/>
     <param name="szOrderType">S4</param>
     <param name="szBusinessUnit">M30</param>
     <param name="mnAddressNumber">4242</param>
   </params>
   <onError abort="yes">
   <callMethod name="F4211ClearWorkFile" app="NetComm" runOnError="yes">
   <params>
     <param name="mnJobNo" idref="1"/>
     <param name="szComputerID" idref="2"/>
     <param name="cClearHeaderWF">2</param>
     <param name="cClearDetailWF">2</param>
   </params>
   </callMethod>
   </onError>
   </callMethod>
   <callMethod name="F4211FSEditLine" app="NetComm" runOnError="yes">
   <params>
     <param name="mnCMJobNo" idref="1"/>
     <param name="szCMComputerID" idref="2"/>
     <param name="szBusinessUnit">M30</param>
     <param name="szItemNo">1001</param>
   </params>
```

```
   <onError abort="no"/>
   </callMethod>
   <callMethod name="F4211FSEndDoc" app="NetComm" runOnError="no">
   <params>
     <param name="mnCMJobNo" idref="1"/>
     <param name="szCMComputerID" idref="2"/>
   </params>
   <onError abort="no">
   <callMethod name="F4211ClearWorkFile" app="NetComm" runOnError="yes">
   <params>
     <param name="mnJobNo" idref="1"/>
     <param name="szComputerID" idref="2"/>
     <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="1"/>
     <param name="szComputerID" idref="2"/>
     <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>
```

I

XML Format Examples (Events)

This appendix contains the following topics:

- Section I.1, "Example: Z Events XML Format"
- Section I.2, "Real-Time Events Template"

I.1 Example: Z Events XML Format

This section illustrates a Z file event XML document.


```
<?xml version='1.0' encoding='utf-8'>
<jdeResponse type='trans' user='JDE' role='*ALL' environment='XDEVNIS2'>
 <transaction type='JDESC' action='transactionInfo'>
  <returnCode code='0'>XML Request OK</returnCode>
  <key>
    <EdiUserId>KW6803955</EdiUserId>
    <EdiBatchNumber>16319</EdiBatchNumber>
    <EdiTransactNumber>106053</EdiTransactNumber>
  </key>
  <F4201Z1 type='header'>
    <EdiUserId>KW6803955</EdiUserId>
    <EdiBatchNumber>16319</EdiBatchNumber>
    <EdiTransactNumber>106053</EdiTransactNumber>
    <EdiLineNumber>1.000</EdiLineNumber>
    <EdiDocumentType>SO</EdiDocumentType>
    <TypeTransaction>JDESC</TypeTransaction>
    <EdiTranslationFormat> </EdiTranslationFormat>
    <EdiTransmissionDate> </EdiTransmissionDate>
    <DirectionIndicator>2</DirectionIndicator>
    <EdiDetailLinesProcess>0</EdiDetailLinesProcess>
    <EdiSuccessfullyProcess>Y</EdiSuccessfullyProcess>
    <TradingPartnerId> </TradingPartnerId>
    <TransactionAction>UA</TransctionAction>
    <CompanyKeyOrderNo>00200</CompanyKeyOrderNo>
    <DocumentOrderInvoiceF>6559</DocumentOrderInvoiceF>
    <OrderType>SO</OrderType>
    <OrderSuffix>000</OrderSuffix>
    <CostCenter>  M30</CostCenter>
    <Company>00200</Company>
    <CompanyKeyOriginal> </CompanyKeyOriginal>
    <OriginalPoSoNumber> </OriginalPoSoNumber>
    <OriginalOrderType> </OriginalOrderType>
    <CompanyKeyRelated> </CompanyKeyRelated>
    <RelatedPoSoNumber> </RelatedPoSoNumber>
    <RelatedOrderType> </RelatedOrderType>
    <AddressNumber>4242</AddressNumber>
```

```
<AddressNumberShipTo>4242</AddressNumberShipTo>
<AddressNumberParent>4242</AddressNumberParent>
<DateRequestedJulian>2005/05/05</DateRequestedJulian>
<DateTransactionJulian>2005/05/05</DateTransactionJulian>
<PromisedDeliveryDate>2005/05/05</PromisedDeliveryDate>
<DateOriginalPromise>2005/05/05</DateOriginalPromise>
<ActualDeliveryDate></ActualDeliveryDate>
<CancelDate></CancelDate>
<DatePriceEffectiveDate>2005/05/05</DatePriceEffectiveDate>
<DatePromisedPickJu>2005/05/05</DatePromisedPickJu>
<DatePromisedShipJu></DatePromisedShipJu>
<Reference1> </Reference1>
<Reference2Vendor> </Reference2Vendor>
<DeliveryInstructLine1> </DeliveryInstructLine1>
<DeliveryInstructLine2> </DeliveryInstructLine2>
<PrintMessage1> </PrintMessage1>
<PaymentTermsCode01> </PaymentTermsCode01>
<PaymentInstrumentA> </PaymentInstrumentA>
<PriceAdjustmentScheduleN>
</PriceAdjustmentScheduleN>
<PricingGroup>PREFER</PricingGroup>
<DiscountTrade>.000</DiscountTrade>
<PercentRetainage1>.000</PercentRetainage1>
<TaxArea1>DEN</TaxArea1>
<TaxExplanationCode1>S</TaxExplanationCode1>
<CertificateTaxExempt> </CertificateTaxExempt>
<AssociatedText> </AssociatedText>
<PriorityProcessing>0</PriorityProcessing>
<BackordersAllowedYN>Y</BackordersAllowedYN>
<SubstitutesAllowedYN>Y</SubstitutesAllowedYN>
<HoldOrdersCode> </HoldOrdersCode>
<PricePickListYN>Y</PricePickListYN>
<InvoiceCopies>0</InvoiceCopies>
<NatureOfTransaction> </NatureOfTransction>
<BuyerNumber>0</BuyerNumber>
<Carrier>0</Carrier>
<ModeOfTransport> </ModeOfTransport>
<ConditionsOfTransport> </ConditionsOfTransport>
<RouteCode> </RouteCode>
<StopCode> </StopCode>
<ZoneNumber> </ZoneNumber>
<ContainerID> </ContainerID>
<FreightHandlingCode> </FreightHandlingcode>
<ApplyFreightYN>Y</ApplyFreightYN>
<ApplyFreight> </ApplyFreight>
<FreightCalculatedYN> </FreightCalculatedYN>
<MergeOrdersYN> </MergeOrdersYN>
<CommissionCode1>6001</CommissionCode1>
<RateCommission1>5.000</RateCommission1>
<CommissionCode2>0</CommissionCode2>
<RateCommission2>.000</RateCommission2>
<ReasonCode> </ReasonCode>
<PostQuantities> </PostQuantities>
<AmountOrderGross>134.97</AmountOrderGross>
<AmountTotalCost>.00</AmountTotalCost>
<UnitOfMeasureWhtDisp> </UnitOfMeasureWhtDisp>
<UnitOfMeasureVolDisp> </UnitOfMeasureVolDisp>
<AuthorizationNoCredit> </AuthorizationNoCredit>
<AcctNoCrBank> </AcctNoCrBank>
<DateExpired></DateExpired>
```

```
  <SubledgerInactiveCode> </SubledgerInactiveCode>
  <CorrespondenceMethod> </CorrespondenceMethod>
  <CurrencyMode>F</CurrencyMode>
  <CurrencyCodeFrom>BEF</CurrencyCodeFrom>
  <CurrencyConverRateOv>33.8180588</CurrencyConverRateOv>
  <LanguagePreference>E</LanguagePreference>
  <AmountForeignOpen>4564.42</AmountForeignOpen
  <AmountForeignTotalC>.00</AmountForeignTotalC>
  <OrderedBy> </OrderedBy>
  <OrderTakenBy> </OrderTakenBy>
  <UserReservedCode> </UserReservedCode>
  <UserReservedDate> </UserReservedDate>
  <UserReservedAmount>.00</UserReservedAmount>
  <UserReservedNumber>0</UserReservedNumber>
  <UserReservedReference> </UserReservedReference>
  <UserId>KW6803955</UserId>
  <ProgramId> </ProgramId>
  <WorkStationId>ST15</WorkStationId>
  <DateUpdated>2000/08/22</DatedUpdated>
  <TimeOfDay>134435</TimeOfDay>
</F4201Z1>
<F4211Z1 type='detail'>
  <EdiUserId>KW6803955</EdiUserId>
  <EdiBatchNumber>16319</EdiBatchNumber>
  <EdiTransactNumber>106053</EdiTransactNumber>
  <EdiLineNumber>1.000</EdiLineNumber>
  <EdiDocumentType>SO</EdiDocumentType>
  <TypeTransaction>JDESC</TypeTransaction>
  <EdiTranslationFormat> </EdiTranslationFormat>
  <EdiTransmissionDate></EdiTransmissionDate>
  <DirectionIndicator>2</DirectionIndicator>
  <EdiDetailLinesProcess>0</EdiDetailLinesProcess>
  <EdiSuccessfullyProcess>N</EdiSuccessfullyProcess>
  <TradingPartnerId> </TradingPartnerId>
  <TransactionAction>UA</TransactionAction>
  <CompanyKeyOrderNo>00200</CompanyKeyOrderNo>
  <DocumentOrderInvoiceE>6559</DocumentOrderInvoiceE>
  <OrderType>SO</OrderType>
  <LineNumber>1.000</LineNumber>
  <OrderSuffix>000</OrderSuffix>
  <CostCenter>  M30</CostCenter>
  <Company>00200</Company>
  <CompanyKeyOriginal> </CompanyKeyOriginal>
  <OriginalPoSoNumber> </OriginalPoSoNumber>
  <OriginalOrderType> </OriginalOrderType>
  <OriginalLineNumber>.000</OriginalLineNumber>
  <CompanyKeyRelated> </CompanyKeyRelated>
  <RelatedPoSoNumber> </RelatedPoSoNumber>
  <RelatedOrderType> </RelatedOrderType>
  <RelatedPoSoLineNo>.000</RelatedPoSoLineNo>
  <ContractNumberDistributi> </ContractNumberDistributi>
  <ContractSupplementDistri>0</ContractSupplementDistri>
  <ContractBalancesUpdatedY> </ContractBalancesUpdatedY>
  <AddressNumber>4242</AddressNumber>
  <AddressNumberShipTo>4242</AddressNumberShipTo>
  <AddressNumberParent>4242</AddressNumberParent>
  <DateRequestedJulian>2005/05/05</DateRequestedJulian>
  <DateTransactionJulian>2005/05/05</DateTransactionJulian>
  <PromisedDeliveryDate>2005/05/05</PromisedDeliveryDate>
  <DateOriginalPromised>2005/05/05</DateOriginalPromised>
```

```
<ActualDeliveryDate></ActualDeliveryDate>
<DateInvoiceJulian></DateInvoiceJulian>
<CancelDate></CancelDate>
<DtForGLAndVouch1></DtForGLAndVouch1>
<DateReleaseJulian>2005/05/05</DateReleaseJulian>
<DatePriceEffectiveDate>2005/05/05</DatePriceEffectiveDate>
<DatePromisedPickJu>2005/05/05</DatePromisedPickJu>
<DatePromisedShipJu></DatePromisedShipJu>
<Reference1> </Reference1>
<Reference2Vendor> </Reference2Vendor>
<IdentifierShortItem>60003</IdentifierShortItem>
<Identifier2ndItem>1001</Identifier2ndItem>
<Identifier3rdItem>1001</Identifier3rdItem>
<Location> </Location>
<Lot> </Lot>
<FromGrade> </FromGrade>
<ThruGrade> </ThruGrade>
<FromPotency>.000</FromPotency>
<ThruPotency>.000</ThruPotency>
<DaysPastExpiration>0</DaysPastExpiration>
<DescriptionLine1>Bike Rack - Trunk Mount</DescriptionLine1>
<DescriptionLine2> </DescriptionLine2>
<LineType>S</LineType>
<StatusCodeNext>540</StatusCodeNext>
<StatusCodeLast>520</StatusCodeLast>
<CostCenterHeader>  M30</CostCenterHeader>
<ItemNumberRelatedKit> </ItemNumberRelatedKit>
<LineNumberKitMaster>.000</LineNumberKitMaster>
<ComponentNumber>.0</ComponentNumber>
<RelatedKitComponent>0</RelatedKitComponent>
<NumbOfCpntPerParent>0</NumbOfCpntPerParent>
<SalesReportingCode1> </SalesReportingCode1>
<SalesReportingCode2> </SalesReportingCode2>
<SalesReportingCode3> </SalesReportingCode3>
<SalesReportingCode4> </SalesReportingCode4>
<SalesReportingCode5> </SalesReportingCode5>
<PurchasingReportCode1> </PurchasingReportCode1>
<PurchasingReportCode2> </PurchasingReportCode2>
<PurchasingReportCode3> </PurchasingReportCode3>
<PurchasingReportCode4> </PurchasingReportCode4>
<PurchasingReportCode5> </PurchasingReportCode5>
<UnitOfMeasureAsInput>EA</UnitOfMeasureAsInput>
<UnitsTransactionQty>3</UnitsTransactionQty>
<UnitsQuantityShipped>3</UnitsQuantityShipped>
<UnitsQuanBackorHeld>0</UnitsQuanBackorHeld>
<UnitsQuantityCanceled>0</UnitsQuantityCanceled>
<UnitsQuantityFuture>0</UnitsQuantityFuture>
<UnitsOpenQuantity>0</UnitsOpenQuantity>
<QuantityShippedToDate>0</QuantityShippedToDate>
<QuantityRelieved>0</QuantityRelieved>
<CommittedHS>S</CommittedHS>
<OtherQuantity12> </OtherQuantity12>
<AmtPricePerUnit2>44.9900</AmtPricePerUnit2>
<AmountExtendedPrice>134.97</AmountExtendedPrice>
<AmountOpen1>.00</AmountOpen1>
<PriceOverrideCode> </PriceOverrideCode>
<TemporaryPriceYN> </TemporaryPriceYN>
<UnitOfMeasureEntUP>EA</UnitOfMeasureEntUP>
<AmtListPricePerUnit>44.9900</AmtListPricePerUnit>
<AmountUnitCost>32.1000</AmountUnitCost>
```

```
<AmountExtendedCost>96.30</AmountExtendedCost>
<CostOverrideCode> </CostOverrideCode>
<ExtendedCostTransfer>.0000</ExtendedCostTransfer>
<PrintMessage1> </PrintMessage1>
<PaymentTermsCode01> </PaymentTermsCode01>
<PaymentInstrumentA> </PaymentInstrumentA>
<BasedonDate> </BasedonDate>
<DiscountTrade>.000</DiscountTrade>
<TradeDiscountOld>.0000</TradeDiscountOld>
<PriceAdjustmentScheduleN> </PriceAdjustmentScheduleN>
<PricingCategory> </PricingCategory>
<PricingCategoryLevel1> </PricingCategoryLevel1>
<DiscountFactor>1.0000</DiscountFactor>
<DiscountFactorTypeOr> </DiscountFactorTypeOr>
<DiscntApplicationType> </DiscntApplicationType>
<DiscountCash>.000</DiscountCash>
<CompanyKey> </CompanyKey>
<DocVoucherInvoiceE>0</DocVoucherInvoiceE>
<DocumentType> </DocumentType>
<OriginalDocumentNo>0</OriginalDocumentNo>
<OriginalDocumentType> </OriginalDocumentType>
<DocumentCompanyOriginal> </DocumentCompanyOriginal>
<PickSlipNumber>0</PickSlipNumber>
<DeliveryNumber>0</DeliveryNumber>
<PromotionNumber> </PromotionNumber>
<DraftNumber>0</DraftNumber>
<TaxableYN>N</TaxableYN>
<TaxArea1>DEN</TaxArea1>
<TaxExplanationCode1>S</TaxExplanationCode1>
<AssociatedText> </AssociatedText>
<PriorityProcessing>0</PriroityProcessing>
<ResolutionCodeBC> </ResolutionCodeBC>
<BackordersAllowedYN>Y</BackordersAllowedYN>
<SubstitutesAllowedYN>Y</SubstitutesAllowedYN>
<PartialShipmentsAllowY>Y</PartialShipmentsAllowY>
<LineofBusiness> </LineofBusiness>
<EndUse> </EndUse>
<DutyStatus> </DutyStatus>
<CommodityCode> </CommodityCode>
<NatureOfTransction> </NatureOfTransaction>
<PrimaryLastVendorNo>4343</PrimaryLastVendorNo>
<BuyerNumber>8444</BuyerNumber>
<Carrier>0</Carrier>
<ModeOfTransport> </ModeOfTransport>
<ConditionsOfTransport> </ConditionsOfTransport>
<RouteCode> </RouteCode>
<StopCode> </StopCode>
<ZoneNumber> </ZoneNumber>
<ContainerID> </ContainerID>
<FreightHandlingCode> </FreightHandlingCode>
<ApplyFreightYN>Y</ApplyFreightYN>
<ApplyFreight> </ApplyFreight>
<FreightCalculatedYN> </FreightCalculatedYN>
<RateCodeFreightMisc> </RateCodeFreightMisc>
<RateTypeFreightMisc> </RateTypeFreightMisc>
<ShippingCommodityClass> </ShippingCommodityClass>
<ShippingConditionsCode> </ShippingConditionsCode>
<SerialNumberLot> </SerialNumberLot>
<UnitOfMeasurePrimary>EA</UnitOfMeasurePrimary>
<UnitsPrimaryQtyOrder>3</UnitsPrimaryQtyOrder>
```

```
<UnitOfMeasureSecondary>EA</UnitOfMeasureSecondary>
<UnitsSecondaryQtyOr>3</UnitsSecondaryQtyOr>
<UnitOfMeasurePricing>EA</UnitOfMeasurePricing>
<AmountUnitWeight>240.0000</AmountUnitWeight>
<WeightUnitOfMeasure>OZ</WeightUnitOfMeasure>
<AmountUnitVolume>6.7500</AmountUnitVolume>
<VolumeUnitOfMeasure>FC</VolumeUnitOfMeasure>
<RepriceBasketPriceCat> </RepriceBasketPriceCat>
<OrderRepriceCategory> </OrderRepriceCategory>
<OrderRepricedIndicator> </OrderRepricedIndicator>
<InventoryCostingMeth>07</InventoryCostingMeth>
<AllocatedByLot> </AllocatedByLot>
<GlClass>IN30</GlClass>
<Century>20</Century>
<FiscalYear1>5</FiscalYear1>
<LineStatus> </LineStatus>
<SalesOrderStatus01> </SalesOrderStatus01>
<SalesOrderStatus02> </SalesOrderStatus02>
<SalesOrderStatus03> </SalesOrderStatus03>
<SalesOrderStatus04> </SalesOrderStatus04>
<SalesOrderStatus05> </SalesOrderStatus05>
<SalesOrderStatus06> </SalesOrderStatus06>
<SalesOrderStatus07> </SalesOrderStatus07>
<SalesOrderStatus08> </SalesOrderStatus08>
<SalesOrderStatus09> </SalesOrderStatus09>
<SalesOrderStatus10> </SalesOrderStatus10>
<SalesOrderStatus11> </SalesOrderStatus11>
<SalesOrderStatus12> </SalesOrderStatus12>
<SalesOrderStatus13> </SalesOrderStatus13>
<SalesOrderStatus14> </SalesOrderStatus14>
<SalesOrderStatus15> </SalesOrderStatus15>
<Salesperson1>6001</Salesperson1>
<SalespersonCommission1>5.000</SalespersonCommission1>
<Salesperson2>0</Salesperson2>
<SalespersonCommission2>.000</SalespersonCommission2>
<ApplyCommissionYN>Y</ApplyCommissionYN>
<CommissionCategory> </CommissionCategory>
<ReasonCode> </ReasonCode>
<GrossWeight>.0000</GrossWeight>
<UnitOfMeasureGrossWt> </UnitOfMeasureGrossWt>
<AcctNoInputMode> </AcctNoInputMode>
<AccountId> </AccountId>
<PurchasingCostCenter> </PurchasingCostCenter>
<ObjectAccount> </ObjectAccount>
<Subsidiary> </Subsidiary>
<LedgerType> </LedgerType>
<Subledger> </Subledger>
<SubledgerType> </SubledgerType>
<CodeLocationTaxStat> </CodeLocationTaxStat>
<PriceCode1> </PriceCode1>
<PriceCode2> </PriceCode2>
<PriceCode3> </PriceCode3>
<StatusInWarehouse> </StatusInWarehouse>
<WoOrderFreezeCode> </WoOrderFreezeCode>
<CorrespondenceMethod> </CorrespondenceMethod>
<CurrencyCodeFrom>BEF</CurrencyCodeFrom>
<CurrencyConverRateOv>33.8180588</CurrencyConverRateOV>
<AmountListPriceForeign>1521.4745</AmountListPriceForeign>
<AmtForPricePerUnit>1521.4745</AmtForPricePerUnit>
<AmountForeignExtPrice>4564.42</AmountForeignExtPrice>
```

```
  <AmountForeignUnitCost>1085.5597</AmountForeignUnitCost>
  <AmountForeignExtCost>3256.68</AmountForeignExtCost>
  <UserReservedCode> </UserReservedCode>
  <UserReservedDate></UserReservedDate>
  <UserReservedAmount>.00</UserReservedAmount>
  <UserReservedNumber>0</UserReservedNumber>
  <UserReservedReference> </UserReservedReference>
  <TransactionOriginator>KW6803955</TransactionOriginator>
  <UserId>KW6803955</UserId>
  <ProgramId>XMLtest</ProgramId>
  <WorkStationId>STI5</WorkStationId>
  <DateUpdated>2000/08/22</DateUpdated>
  <TimeOfDay>134435</TimeOfDay>
</F4211Z1>
<F49211Z1 type='additionalHeader'>
  <EdiUserId>KW6803955</EdiUserId>
  <EdiBatchNumber>16319</EdiBatchNumber>
  <EdiTransactNumber>106053<EdiTransactNumber>
  <EdiLineNumber>1.000</EdiLineNumber>
  <EdiDocumentType>SO</EdiDocumentType>
  <TypeTransaction>JDESC</TypeTransaction>
  <EdiTranslationFormat> </EdiTranslationFormat>
  <EdiTransmissionDate></EdiTransmissionDate>
  <DirectionIndicator>2</DirectionIndicator>
  <EdiDetailLinesProcess>0</EditDetailLinesProcess>
  <EdiSuccessfullyProcess>N</EdiSuccessfullyProcess>
  <TradingPartnerId> </TradingPartnerId>
  <TransactionAction>UA</TransactionAction>
  <DocumentOrderInvoiceE>6559</DocumentOrderInvoiceE>
  <OrderType>SO</OrderType>
  <CompanyKeyOrderNo>00200</CompanyKeyOrderNo>
  <LineNumber>1.000</LineNumber>
  <CostCenterTrip> </CostCenterTrip>
  <TripNumber>0</TripNumber>
  <DateLoaded> </DateLoaded>
  <DispatchGrp> </DispatchGrp>
  <BulkPackedFlag>P</BulkPackedFlag>
  <Distance>0</Distance>
  <UnitOfMeasure> </UnitOfMeasure>
  <DeferredEntriesFlag> </DeferredEntriesFlag>
  <AmountDeferredCost>.0000</AmountDeferredCost>
  <AmountForeignDeferredCos>.0000</AmountForeignDeferredCos>
  <AmountDeferredRevenue>.0000</AmountDeferredRevenue>
  <AmountForeignDeferredRe>.0000</AmountForeignDeferredRe>
  <AaiTableNumber>0</AaiTableNumber>
  <ScheduledInvoiceDate></ScheduledInvoiceDate>
  <InvoiceCycleCode> </InvoiceCycleCode>
  <LoadConfirmDate></LoadConfirmDate>
  <TimeLoad>0</TimeLoad>
  <DeliveryConfirmDate></DelieveryConfirmDate>
  <UnitsPrimaryCommittedQua>0</UnitsPrimaryCommittedQua>
  <UnitofMeasureCommittedQu> <UnitofMeasureCommittedQu>
  <Temperature>.00</Temperature>
  <StrappingTemperatureUnit> </StrappingTemperatureUnit>
  <Density>.00</Density>
  <DensityTypeAtStandardTem> </DensityTypeAtStandardTem>
  <DensityTemperature>.00</DensityTemperature>
  <DensityTemperatureUnit> </DensityTemperatureUnit>
  <VolumeCorrectionFactors>.0000</VolumeCorrectionFactors>
  <PriceatAmbiantorStandard>A</PriceatAmbiantorStandard>
```

```
<PricingBasedOnDate> </PricingBasedOnDate>
<UnitsInvoiceQuantity>0</UnitsInvoiceQuantity>
<StockTotalinPrimaryUOM>0</StockTotalinPrimaryUOM>
<UnitofMeasure6> </UnitofMeasure6>
<AmbientResult>0</AmbientResult>
<UnitofMeasure3> </UnitofMeasure3>
<WeightResult>0</WeightResult>
<UnitofMeasure5> </UnitofMeasure5>
<VendorFreightCalculatedY> </VendorFreightCalculatedY>
<CustomerFreightCalculate> </CustomerFreightCalculate>
<AmountCustomerFreightCha>.0000</AmountCustomerFreightCha>
<AmountVendorFreightCharg>.0000</AmountVendorFreightCharg>
<PrimaryVehicleId> </PrimaryVehicleId>
<RegistrationLicenseNumber> </RegistrationLicenseNumber>
<CostCenterArDefault> </CostCenterArDefault>
<FlightNumber> </FlightNumber>
<Destination> </Destination>
<AircraftType> </AircraftType>
<Origin> /Orign>
<TimeElapsed>0</TimeElapsed>
<ShipmentNumberB73>0</ShipmentNumberB73>
<AddressNumberIssued>6074</AddressNumberIssued>
<PaymentTermsCode01> </PaymentTermsCode01>
<DocVoucherInvoiceF>0</DocVoucherInvoiceF>
<DocumentType> </DocumentType>
<CompanyKey> </CompanyKey>
<CurrencyConverRateOv>-1.0000000</CurrencyConverRateOv>
<CurrencyCodeFrom> </CurrencyCodeFrom>
<TaxArea1>DEN</TaxArea1>
<TaxExplanationCode1> </TaxExplanationCode1>
<ForeignDomesticFlag> </ForeignDomesticFlag>
<FuelingPort> </FuelingPort>
<RegistrationIdentificati> </RegistrationIdentificati>
<DeliveryLocationN> </DeliveryLocationN>
<AuthorizationName> </AuthorizationName>
<NameAlpha> </NameAlpha>
<MeterTicket1> <MeterTicket1>
<UnitsBeginningThroughput>0</UnitsBeginningThroughput>
<ClosingReading1>0</ClosingReading1>
<MeterTicket2> </MeterTicket2>
<UnitsBeginningThroughpu2>0</UnitsBeginningThroughpu2>
<ClosingReading2>0</ClosingReading2>
<MeterTicket3> </MeterTicket3>
<UnitsBeginningThroughpu3>0</UnitsBeginningThroughpu3>
<ClosingReading3>0</ClosingReading3>
<DataArrival></DateArrival>
<TimeArrival>0</TimeArrival>
<DateDeparture></DateDeparture>
<TimeDeparture>0</TimeDeparture>
<DateStartJobJulian></DateStartJobJulian>
<TimeBeginningHHMM>0</TimeBeginningHHMM>
<DateEnding></DateEnding>
<TimeStopHHMM>0</TimeStopHHMM>
<FutureUse01> </FutureUse01>
<FutureUse02> </FutureUse02>
<FutureUse03> </FutureUse03>
<FutureUse04> </FutureUse04>
<FutureUse05> </FutureUse05>
<FutureUseCode> </FutureUseCode>
<FutureUseQuantity>0</FutureUseQuantity>
```

```
    <FutureUseDate></FutureUseDate>
    <FutureUseUnitofMeasure> </FutureUseUnitofMeasure>
    <UserReservedCode> </UserReservedCode>
    <UserReservedDate> </UserReservedDate>
    <UserReservedAmount>00</UserReservedAmount>
    <UserReservedNumber>0<UserReservedNumber>
    <UserReservedReference> </UserReservedReference>
    <TransactionOriginator> </TransactionOriginator>
    <UserId>KW6803955</UserId>
    <ProgramId>XMLtest</ProgramId>
    <WorkStationId>ST15</WorkStationId>
    <DateUpdated>2000/08/22</DateUpdated.
    <TimeOfDay>134435</TimeOfDay>
  </F49211Z1>
 </transaction>
</jdeResponse>
```

I.2 Real-Time Events Template

This section provides an example of the real-time events template. The example template might not correspond to the exact event that your application uses. Your event might include values that are not in the example template.

The event must be described in the jdeResponse type element. The attribute type is always realTimeEvent. The attributes for user and environment always correspond to the user name and environment that generated the event.

```
<?xml version="1.0" encoding="utf-8" ?>
<jdeResponse type="realTimeEvent" user="" role="*ALL"
session="28980548.1019684006" environment="">
<event>
<header>
```

Code for the header information follows. <eventVersion> is always 1.0, <type> corresponds to the event type, <application> corresponds to the application that created the event, and <version> to the version of the application. The <session ID> is unique for every event. The <scope> is the value of the argument scope that was sent to the real-time event API during creation of the event. The <codepage>element is for encoding of the elements. In the sample, utf-8 is used. The remaining header elements are self-explanatory.

```
<eventVersion>1.0</eventVersion>
<type>RTSOOUT</type>
<user />
<application />
<version />
<sessionID />
<environment />
<host />
<sequenceID />
<date />
<time />
<scope />
<codepage>utf-8</codepage>
</header>
```

The body contains details that describe one data structure for each element. The body contains the date of creation, the name of the file that is creating the data structure, time of creation, and the DSTMPL name of the JD Edwards EnterpriseOne data structure. Type is type of partial event (added as an argument to jdeIEO-EventAdd), executionOrder increases in the real generated event from 1 to elementCount, and parameterCount is the number of fields in the data structure. In this example code, there are three data structures: D34A1050C, D4202150C, and D4202150B. Each data structure is followed by detail elements. When you create an event, the element value is the value of the field, for example: <szNameAlpha type=String>ABC</szNameAlpha >

```
<body elementCount="3">
<detail date="" name="" time="" type="" DSTMPL="D34A1050C"
  executionOrder="" parameterCount="25">
<szNameAlpha type="String"/>
<mnParentAddressNumber type="Double"/>
<szSecondItemNumber type="String"/>
<szThirdItemNumber type="String"/>
<cPriorityProcessing type="Character"/>
<cBackOrdersAllowed type="Character"/>
<cOrderShippedFlag type="Character"/>
<cTransferDirectShipFlag type="Character"/>
<cCommitted type="Character"/>
<mnDaysBeforeExpiration type="Double"/>
<szPurchaseCategoryCode1 type="String"/>
<szPurchaseCategoryCode2 type="String"/>
<szPurchaseCategoryCode3 type="String"/>
<szPurchaseCategoryCode4 type="String"/>
<szRelatedOrderNumber type="String"/>
<szRelatedOrderType type="String"/>
<szRelatedOrderKeyCompany type="String"/>
<szPlanningUnitOfMeasure type="String"/>
<mnPlanningQuantity type="Double"/>
<cAPSFlag type="Character"/>
<cAPSSupplyDemandFlag type="Character"/>
<jdDateUpdated type="Date"/>
<mnTimeUpdated type="Double"/>
<szShipComplete type="String"/>
<mnRelatedOrderLineNumber type="Double"/>
</detail>
<detail date="" name="" time="" type="" DSTMPL="D4202150C"
  executionOrder="" parameterCount="94">
<cOrderAction type="Character"/>
<szOrderType type="String"/>
<szOrderCompany type="String"/>
<mnLineNumber type="Double"/>
<szDetailBranchPlant type="String"/>
<mnShipToAddressNumber type="Double"/>
<jdTransactionDate type="Date"/>
<jdRequestedDate type="Date"/>
<jdScheduledPickDate type="Date"/>
<jdPromisedShipDate type="Date"/>
<jdPromisedDeliveryDate type="Date"/>
<jdCancelDate type="Date"/>
<jdPriceEffectiveDate type="Date"/>
<mnQuantityOrdered type="Double"/>
<mnQuantityShipped type="Double"/>
<mnQuantityBackOrdered type="Double"/>
<mnQuantityCanceled type="Double"/>
<szTransactionUnitOfMeasure type="String"/>
<mnUnitPrice type="Double"/>
<mnExtendedPrice type="Double"/>
<mnForeignUnitPrice type="Double"/>
```

```
<mnForeignExtPrice type="Double"/>
<cPriceOverrideCode type="Character"/>
<cTaxableYN type="Character"/>
<szPriceAdjustmentSchedule type="String"/>
<mnDiscountPercentage type="Double"/>
<szPaymentTerms type="String"/>
<cPaymentInstrument type="Character"/>
<szCurrencyCode type="String"/>
<szItemNumber type="String"/>
<mnShortItemNumber type="Double"/>
<szDescriptionLine1 type="String"/>
<szDescriptionLine2 type="String"/>
<szLineType type="String"/>
<szLastStatus type="String"/>
<szNextStatus type="String"/>
<szLocation type="String"/>
<szLot type="String"/>
<szLineofBusiness type="String"/>
<szEndUse type="String"/>
<szDutyStatus type="String"/>
<szPrintMessage1 type="String"/>
<szFreightHandlingCode type="String"/>
<mnItemWeight type="Double"/>
<szWeightUnitOfMeasure type="String"/>
<szModeOfTransport type="String"/>
<mnCarrier type="Double"/>
<szSubledger type="String"/>
<cSubledgerType type="Character"/>
<szPriceCode1 type="String"/>
<szPriceCode2 type="String"/>
<szPriceCode3 type="String"/>
<szSalesReportingCode1 type="String"/>
<szSalesReportingCode2 type="String"/>
<szSalesReportingCode3 type="String"/>
<szSalesReportingCode4 type="String"/>
<szSalesReportingCode5 type="String"/>
<szOriginalPoSoNumber type="String"/>
<szOriginalOrderType type="String"/>
<szOriginalOrderCompany type="String"/>
<mnOriginalOrderLineNumber type="Double"/>
<jdDateUpdated type="Date"/>
<mnTimeOfDay type="Double"/>
<mnPickSlipNumber type="Double"/>
<mnInvoiceDocNumber type="Double"/>
<szInvoiceDocType type="String"/>
<szInvoceDocCompany type="String"/>
<szUserReservedCode type="String"/>
<jdUserReservedDate type="Date"/>
<mnUserReservedNumber type="Double"/>
<mnUserReservedAmount type="Double"/>
<szUserReservedReference type="String"/>
<mnUnitCost type="Double"/>
<mnExtendedCost type="Double"/>
<mnForeignUnitCost type="Double"/>
<mnForeignExtCost type="Double"/>
<mnOrderNumber type="Double"/>
<szSupplierReference type="String"/>
<jdOriginalPromisdDate type="Date"/>
<mnAdjustmentRevisionLevel type="Double"/>
<mnLastIndex type="Double"/>
```

```
<szRelatedPoSoNumber type="String"/>
<szRelatedOrderType type="String"/>
<szRelatedOrderCompany type="String"/>
<mnRelatedPoSoLineNo type="Double"/>
<szPricingUnitOfMeasure type="String"/>
<szTaxArea type="String"/>
<szTaxExplanationCode type="String"/>
<szPartnerItemNo type="String"/>
<szCatalogItem type="String"/>
<szUPCNumber type="String"/>
<szShipToDescriptive type="String"/>
<szSoldToDescriptive type="String"/>
<szProductItem type="String"/>
</detail>
<detail date="" name="" time="" type="" DSTMPL="D4202150B"
  executionOrder="" parameterCount="66">
<cOrderAction type="Character"/>
<mnOrderNumber type="Double"/>
<szOrderType type="String"/>
<szOrderCompany type="String"/>
<szHeaderBranchPlant type="String"/>
<szCompany type="String"/>
<szOriginalPoSoNumber type="String"/>
<szOrderedBy type="String"/>
<szOrderTakenBy type="String"/>
<mnSoldToAddressNumber type="Double"/>
<szSoldToNameMailing type="String"/>
<szSoldToAddressLine1 type="String"/>
<szSoldToAddressLine2 type="String"/>
<szSoldToAddressLine3 type="String"/>
<szSoldToAddressLine4 type="String"/>
<szSoldToZipCode type="String"/>
<szSoldToCity type="String"/>
<szSoldToCounty type="String"/>
<szSoldToState type="String"/>
<szSoldToCountry type="String"/>
<mnShipToAddressNumber type="Double"/>
<szShipToNameMailing type="String"/>
<szShipToAddressLine1 type="String"/>
<szShipToAddressLine2 type="String"/>
<szShipToAddressLine3 type="String"/>
<szShipToAddressLine4 type="String"/>
<szShipToZipCode type="String"/>
<szShipToCity type="String"/>
<szShipToCounty type="String"/>
<szShipToState type="String"/>
<szShipToCountry type="String"/>
<jdTransactionDate type="Date"/>
<jdRequestedDate type="Date"/>
<jdCancelDate type="Date"/>
<szReference type="String"/>
<szDeliveryInstructLine1 type="String"/>
<szDeliveryInstructLine2 type="String"/>
<szPrintMessage type="String"/>
<szFreightHandlingCode type="String"/>
<mnCommissionCode1 type="Double"/>
<mnCommissionCode2 type="Double"/>
<mnRateCommission1 type="Double"/>
<mnRateCommission2 type="Double"/>
<mnDiscountTrade type="Double"/>
```

```
<szPaymentTerms type="String"/>
<cPaymentInstrument type="Character"/>
<szCurrencyCode type="String"/>
<mnCurrencyConverRate type="Double"/>
<szTaxArea type="String"/>
<szTaxExplanationCode type="String"/>
<mnOrderTotal type="Double"/>
<mnForeignOrderTotal type="Double"/>
<szUserReservedCode type="String"/>
<jdUserReservedDate type="Date"/>
<mnUserReservedAmount type="Double"/>
<mnUserReservedNumber type="Double"/>
<szUserReservedReference type="String"/>
<szHoldCode type="String"/>
<cQuoteFlag type="Character"/>
<jdScheduledPickDate type="Date"/>
<jdPromisedShipDate type="Date"/>
<jdOriginalPromisdDate type="Date"/>
<cCurrencyMode type="Character"/>
<szShipToDescriptive type="String"/>
<szSoldToDescriptive type="String"/>
<cPublishToXPIxFlag type="Character"/>
</detail>
</body>
</event>
</jdeResponse>
```

This table shows the mapping between JD Edwards EnterpriseOne types and events:

JD Edwards EnterpriseOne	Event
CHAR	Character
STRING	String
MATH_numeric	Double
JDEDATE	Dat
SHORT	Int
INT	Int
USHORT	Int
LONG	Long
ULONG	Long
ID	Long
ID2	Long
BOOL	BOOL

Glossary

Accessor Methods/Assessors

Java methods to “get” and “set” the elements of a value object or other source file.

activity rule

The criteria by which an object progresses from one given point to the next in a flow.

add mode

A condition of a form that enables users to input data.

Advanced Planning Agent (APAg)

A JD Edwards EnterpriseOne tool that can be used to extract, transform, and load enterprise data. APAg supports access to data sources in the form of rational databases, flat file format, and other data or message encoding, such as XML.

application server

Software that provides the business logic for an application program in a distributed environment. The servers can be Oracle Application Server (OAS) or WebSphere Application Server (WAS).

Auto Commit Transaction

A database connection through which all database operations are immediately written to the database.

batch processing

A process of transferring records from a third-party system to JD Edwards EnterpriseOne.

In JD Edwards EnterpriseOne Financial Management, batch processing enables you to transfer invoices and vouchers that are entered in a system other than JD Edwards EnterpriseOne to JD Edwards EnterpriseOne Accounts Receivable and JD Edwards EnterpriseOne Accounts Payable, respectively. In addition, you can transfer address book information, including customer and supplier records, to JD Edwards EnterpriseOne.

batch server

A server that is designated for running batch processing requests. A batch server typically does not contain a database nor does it run interactive applications.

batch-of-one

A transaction method that enables a client application to perform work on a client workstation, then submit the work all at once to a server application for further processing. As a batch process is running on the server, the client application can continue performing other tasks.

best practices

Non-mandatory guidelines that help the developer make better design decisions.

BPEL

Abbreviation for Business Process Execution Language, a standard web services orchestration language, which enables you to assemble discrete services into an end-to-end process flow.

BPEL PM

Abbreviation for Business Process Execution Language Process Manager, a comprehensive infrastructure for creating, deploying, and managing BPEL business processes.

Build Configuration File

Configurable settings in a text file that are used by a build program to generate ANT scripts. ANT is a software tool used for automating build processes. These scripts build published business services.

build engineer

An actor that is responsible for building, mastering, and packaging artifacts. Some build engineers are responsible for building application artifacts, and some are responsible for building foundation artifacts.

Build Program

A WIN32 executable that reads build configuration files and generates an ANT script for building published business services.

business analyst

An actor that determines if and why an EnterpriseOne business service needs to be developed.

business function

A named set of user-created, reusable business rules and logs that can be called through event rules. Business functions can run a transaction or a subset of a transaction (check inventory, issue work orders, and so on). Business functions also contain the application programming interfaces (APIs) that enable them to be called from a form, a database trigger, or a non-JD Edwards EnterpriseOne application. Business functions can be combined with other business functions, forms, event rules, and other components to make up an application. Business functions can be created through event rules or third-generation languages, such as C. Examples of business functions include Credit Check and Item Availability.

business function event rule

See named event rule (NER).

business service

EnterpriseOne business logic written in Java. A business service is a collection of one or more artifacts. Unless specified otherwise, a business service implies both a published business service and business service.

business service artifacts

Source files, descriptors, and so on that are managed for business service development and are needed for the business service build process.

business service class method

A method that accesses resources provided by the business service framework.

business service configuration files

Configuration files include, but are not limited to, interop.ini, JDBj.ini, and jdelog.properties.

business service cross reference

A key and value data pair used during orchestration. Collectively refers to both the code and the key cross reference in the WSG/XPI based system.

business service cross-reference utilities

Utility services installed in a BPEL/ESB environment that are used to access JD Edwards EnterpriseOne orchestration cross-reference data.

business service development environment

A framework needed by an integration developer to develop and manage business services.

business services development tool

Otherwise known as JDeveloper.

business service EnterpriseOne object

A collection of artifacts managed by EnterpriseOne LCM tools. Named and represented within EnterpriseOne LCM similarly to other EnterpriseOne objects like tables, views, forms, and so on.

business service framework

Parts of the business service foundation that are specifically for supporting business service development.

business service payload

An object that is passed between an enterprise server and a business services server. The business service payload contains the input to the business service when passed to the business services server. The business service payload contains the results from the business service when passed to the Enterprise Server. In the case of notifications, the return business service payload contains the acknowledgement.

business service property

Key value data pairs used to control the behavior or functionality of business services.

Business Service Property Admin Tool

An EnterpriseOne application for developers and administrators to manage business service property records.

business service property business service group

A classification for business service property at the business service level. This is generally a business service name. A business service level contains one or more business service property groups. Each business service property group may contain zero or more business service property records.

business service property key

A unique name that identifies the business service property globally in the system.

business service property utilities

A utility API used in business service development to access EnterpriseOne business service property data.

business service property value

A value for a business service property.

business service repository

A source management system, for example ClearCase, where business service artifacts and build files are stored. Or, a physical directory in network.

business services server

The physical machine where the business services are located. Business services are run on an application server instance.

business services source file or business service class

One type of business service artifact. A text file with the .java file type written to be compiled by a Java compiler.

business service value object template

The structural representation of a business service value object used in a C-business function.

Business Service Value Object Template Utility

A utility used to create a business service value object template from a business service value object.

business services server artifact

The object to be deployed to the business services server.

business view

A means for selecting specific columns from one or more JD Edwards EnterpriseOne application tables whose data is used in an application or report. A business view does not select specific rows, nor does it contain any actual data. It is strictly a view through which you can manipulate data.

central objects merge

A process that blends a customer's modifications to the objects in a current release with objects in a new release.

central server

A server that has been designated to contain the originally installed version of the software (central objects) for deployment to client computers. In a typical JD Edwards EnterpriseOne installation, the software is loaded on to one machine—the central server. Then, copies of the software are pushed out or downloaded to various workstations attached to it. That way, if the software is altered or corrupted through its use on workstations, an original set of objects (central objects) is always available on the central server.

charts

Tables of information in JD Edwards EnterpriseOne that appear on forms in the software.

check-in repository

A repository for developers to check in and check out business service artifacts. There are multiple check-in repositories. Each can be used for a different purpose (for example, development, production, testing, and so on).

checksum

A fixed-size datum computed from an arbitrary block of digital data for the purpose of detecting accidental errors that may have been introduced during its transmission or storage. JD Edwards EnterpriseOne uses the checksum to verify the integrity of packages that have been downloaded by recomputing the checksum of the downloaded package and comparing it with the checksum of the original package. The procedure that yields the checksum from the data is called a checksum function or checksum algorithm. JD Edwards EnterpriseOne uses the MD5 and STA-1 checksum algorithms.

connector

Component-based interoperability model that enables third-party applications and JD Edwards EnterpriseOne to share logic and data. The JD Edwards EnterpriseOne connector architecture includes Java and COM connectors.

Control Table Workbench

An application that, during the Installation Workbench processing, runs the batch applications for the planned merges that update the data dictionary, user-defined codes, menus, and user override tables.

control tables merge

A process that blends a customer's modifications to the control tables with the data that accompanies a new release.

correlation data

The data used to tie HTTP responses with requests that consist of business service name and method.

credentials

A valid set of JD Edwards EnterpriseOne username/password/environment/role, EnterpriseOne session, or EnterpriseOne token.

cross-reference utility services

Utility services installed in a BPEL/ESB environment that access EnterpriseOne cross-reference data.

database credentials

A valid database username/password.

database server

A server in a local area network that maintains a database and performs searches for client computers.

Data Source Workbench

An application that, during the Installation Workbench process, copies all data sources that are defined in the installation plan from the Data Source Master and Table and Data Source Sizing tables in the Planner data source to the system-release number data source. It also updates the Data Source Plan detail record to reflect completion.

deployment artifacts

Artifacts that are needed for the deployment process, such as servers, ports, and such.

deployment server

A server that is used to install, maintain, and distribute software to one or more enterprise servers and client workstations.

direct connect

A transaction method in which a client application communicates interactively and directly with a server application.

See also batch-of-one and store-and-forward.

Do Not Translate (DNT)

A type of data source that must exist on the iSeries because of BLOB restrictions.

embedded application server instance

An OC4J instance started by and running wholly within JDeveloper.

edit code

A code that indicates how a specific value for a report or a form should appear or be formatted. The default edit codes that pertain to reporting require particular attention because they account for a substantial amount of information.

edit mode

A condition of a form that enables users to change data.

edit rule

A method used for formatting and validating user entries against a predefined rule or set of rules.

Electronic Data Interchange (EDI)

An interoperability model that enables paperless computer-to-computer exchange of business transactions between JD Edwards EnterpriseOne and third-party systems. Companies that use EDI must have translator software to convert data from the EDI standard format to the formats of their computer systems.

embedded event rule

An event rule that is specific to a particular table or application. Examples include form-to-form calls, hiding a field based on a processing option value, and calling a business function. Contrast with the business function event rule.

Employee Work Center

A central location for sending and receiving all JD Edwards EnterpriseOne messages (system and user generated), regardless of the originating application or user. Each user has a mailbox that contains workflow and other messages, including Active Messages.

enterprise server

A server that contains the database and the logic for JD Edwards EnterpriseOne.

Enterprise Service Bus (ESB)

Middleware infrastructure products or technologies based on web services standards that enable a service-oriented architecture using an event-driven and XML-based messaging framework (the bus).

EnterpriseOne administrator

An actor responsible for the EnterpriseOne administration system.

EnterpriseOne credentials

A user ID, password, environment, and role used to validate a user of EnterpriseOne.

EnterpriseOne development client

Historically called “fat client,” a collection of installed EnterpriseOne components required to develop EnterpriseOne artifacts, including the Microsoft Windows client and design tools.

EnterpriseOne extension

A JDeveloper component (plug-in) specific to EnterpriseOne. A JDeveloper wizard

is a specific example of an extension.

EnterpriseOne object

A reusable piece of code that is used to build applications. Object types include tables, forms, business functions, data dictionary items, batch processes, business views, event rules, versions, data structures, and media objects.

EnterpriseOne process

A software process that enables JD Edwards EnterpriseOne clients and servers to handle processing requests and run transactions. A client runs one process, and servers can have multiple instances of a process. JD Edwards EnterpriseOne processes can also be dedicated to specific tasks (for example, workflow messages and data replication) to ensure that critical processes don't have to wait if the server is particularly busy.

EnterpriseOne resource

Any EnterpriseOne table, metadata, business function, dictionary information, or other information restricted to authorized users.

Environment Workbench

An application that, during the Installation Workbench process, copies the environment information and Object Configuration Manager tables for each environment from the Planner data source to the system-release number data source. It also updates the Environment Plan detail record to reflect completion.

escalation monitor

A batch process that monitors pending requests or activities and restarts or forwards them to the next step or user after they have been inactive for a specified amount of time.

event rule

A logic statement that instructs the system to perform one or more operations based on an activity that can occur in a specific application, such as entering a form or exiting a field.

explicit transaction

Transaction used by a business service developer to explicitly control the type (auto or manual) and the scope of transaction boundaries within a business service.

exposed method or value object

Published business service source files or parts of published business service source files that are part of the published interface. These are part of the contract with the customer.

fast path

A command prompt that enables the user to move quickly among menus and applications by using specific commands.

file server

A server that stores files to be accessed by other computers on the network. Unlike a disk server, which appears to the user as a remote disk drive, a file server is a sophisticated device that not only stores files, but also manages them and maintains order as network users request files and make changes to these files.

final mode

The report processing mode of a processing mode of a program that updates or creates data records.

foundation

A framework that must be accessible for execution of business services at runtime. This includes, but is not limited to, the Java Connector and JDBj.

FTP server

A server that responds to requests for files via file transfer protocol.

HTTP Adapter

A generic set of services that are used to do the basic HTTP operations, such as GET, POST, PUT, DELETE, TRACE, HEAD, and OPTIONS with the provided URL.

instantiate

A Java term meaning “to create.” When a class is instantiated, a new instance

is created.

integration developer

The user of the system who develops, runs, and debugs the EnterpriseOne business services. The integration developer uses the EnterpriseOne business services to develop these components.

integration point (IP)

The business logic in previous implementations of EnterpriseOne that exposes a document level interface. This type of logic used to be called XBPs. In EnterpriseOne 8.11, IPs are implemented in Web Services Gateway powered by webMethods.

integration server

A server that facilitates interaction between diverse operating systems and applications across internal and external networked computer systems.

integrity test

A process used to supplement a company's internal balancing procedures by locating and reporting balancing problems and data inconsistencies.

interface table

See Z table.

internal method or value object

Business service source files or parts of business service source files that are not part of the published interface. These could be private or protected methods. These could be value objects not used in published methods.

interoperability model

A method for third-party systems to connect to or access JD Edwards EnterpriseOne.

in-your-face error

In JD Edwards EnterpriseOne, a form-level property which, when enabled, causes the text of application errors to appear on the form.

jargon

An alternative data dictionary item description that JD Edwards EnterpriseOne appears based on the product code of the current object.

Java application server

A component-based server that resides in the middle-tier of a server-centric architecture. This server provides middleware services for security and state maintenance, along with data access and persistence.

JDBNET

A database driver that enables heterogeneous servers to access each other's data.

JDEBASE Database Middleware

A JD Edwards EnterpriseOne proprietary database middleware package that provides platform-independent APIs, along with client-to-server access.

JDECallObject

An API used by business functions to invoke other business functions.

jde.ini

A JD Edwards EnterpriseOne file (or member for iSeries) that provides the runtime settings required for JD Edwards EnterpriseOne initialization. Specific versions of the file or member must reside on every machine running JD Edwards EnterpriseOne. This includes workstations and servers.

JDEIPC

Communications programming tools used by server code to regulate access to the same data in multiprocess environments, communicate and coordinate between processes, and create new processes.

jde.log

The main diagnostic log file of JD Edwards EnterpriseOne. This file is always located in the root directory on the primary drive and contains status and error messages from the startup and operation of JD Edwards EnterpriseOne.

JDENET

A JD Edwards EnterpriseOne proprietary communications middleware package. This package is a peer-to-peer, message-based, socket-based, multiprocess communications middleware solution. It handles client-to-server and server-to-server communications for all JD Edwards EnterpriseOne supported platforms.

JDeveloper Project

An artifact that JDeveloper uses to categorize and compile source files.

JDeveloper Workspace

An artifact that JDeveloper uses to organize project files. It contains one or more project files.

JMS Queue

A Java Messaging service queue used for point-to-point messaging.

listener service

A listener that listens for XML messages over HTTP.

local repository

A developer's local development environment that is used to store business service artifacts.

Location Workbench

An application that, during the Installation Workbench process, copies all locations that are defined in the installation plan from the Location Master table in the Planner data source to the system data source.

logic server

A server in a distributed network that provides the business logic for an application program. In a typical configuration, pristine objects are replicated on to the logic server from the central server. The logic server, in conjunction with workstations, actually performs the processing required when JD Edwards EnterpriseOne software runs.

MailMerge Workbench

An application that merges Microsoft Word 6.0 (or higher) word-processing documents with JD Edwards EnterpriseOne records to automatically print business documents. You can use MailMerge Workbench to print documents, such as form letters about verification of employment.

Manual Commit transaction

A database connection where all database operations delay writing to the database until a call to commit is made.

master business function (MBF)

An interactive master file that serves as a central location for adding, changing, and updating information in a database. Master business functions pass information between data entry forms and the appropriate tables. These master functions provide a common set of functions that contain all of the necessary default and editing rules for related programs. MBFs contain logic that ensures the integrity of adding, updating, and deleting information from databases.

master table

See published table.

media storage object

Files that use one of the following naming conventions that are not organized into table format: Gxxx, xxxGT, or GTxxx.

message center

A central location for sending and receiving all JD Edwards EnterpriseOne messages (system and user generated), regardless of the originating application or user.

messaging adapter

An interoperability model that enables third-party systems to connect to JD Edwards EnterpriseOne to exchange information through the use of messaging queues.

messaging server

A server that handles messages that are sent for use by other programs using a messaging API. Messaging servers typically employ a middleware program to perform their functions.

Monitoring Application

An EnterpriseOne tool provided for an administrator to get statistical information for various EnterpriseOne servers, reset statistics, and set notifications.

named event rule (NER)

Encapsulated, reusable business logic created using event rules, rather that C programming. NERs are also called business function event rules. NERs can be reused in multiple places by multiple programs. This modularity lends itself to streamlining, reusability of code, and less work.

Object Configuration Manager (OCM)

In JD Edwards EnterpriseOne, the object request broker and control center for the runtime environment. OCM keeps track of the runtime locations for business functions, data, and batch applications. When one of these objects is called, OCM directs access to it using defaults and overrides for a given environment and user.

Object Librarian

A repository of all versions, applications, and business functions reusable in building applications. Object Librarian provides check-out and check-incapabilities for developers, and it controls the creation, modification, and use of JD Edwards EnterpriseOne objects. Object Librarian supports multiple environments (such as production and development) and enables objects to be easily moved from one environment to another.

Object Librarian merge

A process that blends any modifications to the Object Librarian in a previous release into the Object Librarian in a new release.

Open Data Access (ODA)

An interoperability model that enables you to use SQL statements to extract JD Edwards EnterpriseOne data for summarization and report generation.

Output Stream Access (OSA)

An interoperability model that enables you to set up an interface for JD Edwards EnterpriseOne to pass data to another software package, such as Microsoft Excel, for processing.

package

JD Edwards EnterpriseOne objects are installed to workstations in packages from the deployment server. A package can be compared to a bill of material or kit that indicates the necessary objects for that workstation and where on the deployment server the installation program can find them. It is point-in-time snapshot of the central objects on the deployment server.

package build

A software application that facilitates the deployment of software changes and new applications to existing users. Additionally, in JD Edwards EnterpriseOne, a package build can be a compiled version of the software. When you upgrade your version of the ERP software, for example, you are said to take a package build.

Consider the following context: “Also, do not transfer business functions into the production path code until you are ready to deploy, because a global build of business functions done during a package build will automatically include the new functions.” The process of creating a package build is often referred to, as it is in this example, simply as “a package build.”

package location

The directory structure location for the package and its set of replicated objects. This is usually \\deployment server\release\path_code\package\package name. The subdirectories under this path are where the replicated objects for the package are placed. This is also referred to as where the package is built or stored.

Package Workbench

An application that, during the Installation Workbench process, transfers the package information tables from the Planner data source to the system-release number data source. It also updates the Package Plan detail record to reflect completion.

Pathcode Directory

The specific portion of the file system on the EnterpriseOne development client where EnterpriseOne development artifacts are stored.

patterns

General repeatable solutions to a commonly occurring problem in software design. For business service development, the focus is on the object relationships and interactions.

For orchestrations, the focus is on the integration patterns (for example, synchronous and asynchronous request/response, publish, notify, and receive/reply).

print server

The interface between a printer and a network that enables network clients to connect to the printer and send their print jobs to it. A print server can be a computer, separate hardware device, or even hardware that resides inside of the printer itself.

pristine environment

A JD Edwards EnterpriseOne environment used to test unaltered objects with JD Edwards EnterpriseOne demonstration data or for training classes. You must have this environment so that you can compare pristine objects that you modify.

processing option

A data structure that enables users to supply parameters that regulate the running of a batch program or report. For example, you can use processing options to specify default values for certain fields, to determine how information appears or is printed, to specify date ranges, to supply runtime values that regulate program execution, and so on.

production environment

A JD Edwards EnterpriseOne environment in which users operate EnterpriseOne software.

Production Published Business Services Web Service

Published business services web service deployed to a production application server.

program temporary fix (PTF)

A representation of changes to JD Edwards EnterpriseOne software that your organization receives on magnetic tapes or disks.

project

In JD Edwards EnterpriseOne, a virtual container for objects being developed in Object Management Workbench.

promotion path

The designated path for advancing objects or projects in a workflow. The following is the normal promotion cycle (path):

11>21>26>28>38>01

In this path, 11 equals new project pending review, 21 equals programming, 26 equals QA test/review, 28 equals QA test/review complete, 38 equals in production, 01 equals complete. During the normal project promotion cycle, developers check objects out of and into the development path code and then promote them to the prototype path code. The objects are then moved to the productions path code before declaring them complete.

proxy server

A server that acts as a barrier between a workstation and the internet so that the enterprise can ensure security, administrative control, and caching service.

published business service

EnterpriseOne service level logic and interface. A classification of a published business service indicating the intention to be exposed to external (non-EnterpriseOne) systems.

published business service identification information

Information about a published business service used to determine relevant authorization records. Published business services + method name, published business services, or *ALL.

published business service web service

Published business services components packaged as J2EE Web Service (namely, a J2EE EAR file that contains business service classes, business service foundation, configuration files, and web service artifacts).

published table

Also called a master table, this is the central copy to be replicated to other machines. Residing on the publisher machine, the F98DRPUB table identifies all of the published tables and their associated publishers in the enterprise.

publisher

The server that is responsible for the published table. The F98DRPUB table identifies all of the published tables and their associated publishers in the enterprise.

QBE

An abbreviation for query by example. In JD Edwards EnterpriseOne, the QBE line is the top line on a detail area that is used for filtering data.

real-time event

A message triggered from EnterpriseOne application logic that is intended for external systems to consume.

refresh

A function used to modify JD Edwards EnterpriseOne software, or subset of it, such as a table or business data, so that it functions at a new release or cumulative update level.

replication server

A server that is responsible for replicating central objects to client machines.

rules

Mandatory guidelines that are not enforced by tooling, but must be followed in order to accomplish the desired results and to meet specified standards.

secure by default

A security model that assumes that a user does not have permission to execute an object unless there is a specific record indicating such permissions.

Secure Socket Layer (SSL)

A security protocol that provides communication privacy. SSL enables client and server applications to communicate in a way that is designed to prevent eavesdropping, tampering, and message forgery.

selection

Found on JD Edwards EnterpriseOne menus, a selection represents functions that you can access from a menu. To make a selection, type the associated number in the Selection field and press Enter.

serialize

The process of converting an object or data into a format for storage or transmission across a network connection link with the ability to reconstruct the original data or objects when needed.

Server Workbench

An application that, during the Installation Workbench process, copies the server configuration files from the Planner data source to the system-release number data source. The application also updates the Server Plan detail record to reflect completion.

SOA

Abbreviation for Service Oriented Architecture.

softcoding

A coding technique that enables an administrator to manipulate site-specific variables that affect the execution of a given process.

source repository

A repository for HTTP adapter and listener service development environment artifacts.

Specification merge

A merge that comprises three merges: Object Librarian merge, Versions List merge, and Central Objects merge. The merges blend customer modifications with data that accompanies a new release.

specification

A complete description of a JD Edwards EnterpriseOne object. Each object has its own specification, or name, which is used to build applications.

Specification Table Merge Workbench

An application that, during the Installation Workbench process, runs the batch applications that update the specification tables.

SSL Certificate

A special message signed by a certificate authority that contains the name of a user and that user's public key in such a way that anyone can "verify" that the message was signed by no one other than the certification authority and thereby develop trust in the user's public key.

store-and-forward

The mode of processing that enables users who are disconnected from a server to enter transactions and then later connect to the server to upload those transactions.

subscriber table

Table F98DRSUB, which is stored on the publisher server with the F98DRPUB table and identifies all of the subscriber machines for each published table.

super class

An inheritance concept of the Java language where a class is an instance of something, but is also more specific. “Tree” might be the super class of “Oak” and “Elm,” for example.

table access management (TAM)

The JD Edwards EnterpriseOne component that handles the storage and retrieval of use-defined data. TAM stores information, such as data dictionary definitions; application and report specifications; event rules; table definitions; business function input parameters and library information; and data structure definitions for running applications, reports, and business functions.

Table Conversion Workbench

An interoperability model that enables the exchange of information between JD Edwards EnterpriseOne and third-party systems using non-JD Edwards EnterpriseOne tables.

table conversion

An interoperability model that enables the exchange of information between JD Edwards EnterpriseOne and third-party systems using non-JD Edwards EnterpriseOne tables.

table event rules

Logic that is attached to database triggers that runs whenever the action specified by the trigger occurs against the table. Although JD Edwards EnterpriseOne enables event rules to be attached to application events, this functionality is application specific. Table event rules provide embedded logic at the table level.

terminal server

A server that enables terminals, microcomputers, and other devices to connect to a network or host computer or to devices attached to that particular computer.

transaction processing (TP) monitor

A monitor that controls data transfer between local and remote terminals and the applications that originated them. TP monitors also protect data integrity in the distributed environment and may include programs that validate data and format terminal screens.

transaction processing method

A method related to the management of a manual commit transaction boundary (for example, start, commit, rollback, and cancel).

transaction set

An electronic business transaction (electronic data interchange standard document) made up of segments.

trigger

One of several events specific to data dictionary items. You can attach logic to a data dictionary item that the system processes automatically when the event occurs.

triggering event

A specific workflow event that requires special action or has defined consequences or resulting actions.

user identification information

User ID, role, or *public.

User Overrides merge

Adds new user override records into a customer's user override table.

value object

A specific type of source file that holds input or output data, much like a data structure passes data. Value objects can be exposed (used in a published business service) or internal, and input or output. They are comprised of simple and complex elements and accessories to those elements.

versioning a published business service

Adding additional functionality/interfaces to the published business services without modifying the existing functionality/interfaces.

Versions List merge

The Versions List merge preserves any non-XJDE and non-ZJDE version specifications for objects that are valid in the new release, as well as their processing options data.

visual assist

Forms that can be invoked from a control via a trigger to assist the user in determining what data belongs in the control.

vocabulary override

An alternate description for a data dictionary item that appears on a specific JD Edwards EnterpriseOne form or report.

web application server

A web server that enables web applications to exchange data with the back-end systems and databases used in eBusiness transactions.

web server

A server that sends information as requested by a browser, using the TCP/IP set of protocols. A web server can do more than just coordination of requests from browsers; it can do anything a normal server can do, such as house applications or data. Any computer can be turned into a web server by installing server software and connecting the machine to the internet.

Web Service Description Language (WSDL)

An XML format for describing network services.

Web Service Inspection Language (WSIL)

An XML format for assisting in the inspection of a site for available services and a set of rules for how inspection-related information should be made.

web service softcoding record

An XML document that contains values that are used to configure a web service proxy. This document identifies the endpoint and conditionally includes security information.

web service softcoding template

An XML document that provides the structure for a soft coded record.

Where clause

The portion of a database operation that specifies which records the database operation will affect.

Windows terminal server

A multiuser server that enables terminals and minimally configured computers to display Windows applications even if they are not capable of running Windows software themselves. All client processing is performed centrally at the Windows terminal server and only display, keystroke, and mouse commands are transmitted over the network to the client terminal device.

wizard

A type of JDeveloper extension used to walk the user through a series of steps.

workbench

A program that enables users to access a group of related programs from a single entry point. Typically, the programs that you access from a workbench are used to complete a large business process. For example, you use the JD Edwards EnterpriseOne Payroll Cycle Workbench (P07210) to access all of the programs that the system uses to process payroll, print payments, create payroll reports, create journal entries, and update payroll history. Examples of JD Edwards EnterpriseOne workbenches include Service Management Workbench (P90CD020), Line Scheduling Workbench (P3153), Planning Workbench (P13700), Auditor's Workbench (P09E115), and Payroll Cycle Workbench.

workflow

The automation of a business process, in whole or in part, during which documents, information, or tasks are passed from one participant to another for action, according to a set of procedural rules.

workgroup server

A server that usually contains subsets of data replicated from a master network server. A workgroup server does not perform application or batch processing.

XAPI events

A service that uses system calls to capture JD Edwards EnterpriseOne transactions as they occur and then calls third-party software, end users, and other JD Edwards EnterpriseOne systems that have requested notification when the specified transactions occur to return a response.

XML CallObject

An interoperability capability that enables you to call business functions.

XML Dispatch

An interoperability capability that provides a single point of entry for all XML documents coming into JD Edwards EnterpriseOne for responses.

XML List

An interoperability capability that enables you to request and receive JD Edwards EnterpriseOne database information in chunks.

XML Service

An interoperability capability that enables you to request events from one JD Edwards EnterpriseOne system and receive a response from another JD Edwards EnterpriseOne system.

XML Transaction

An interoperability capability that enables you to use a predefined transaction type to send information to or request information from JD Edwards EnterpriseOne. XML transaction uses interface table functionality.

XML Transaction Service (XTS)

Transforms an XML document that is not in the JD Edwards EnterpriseOne format into an XML document that can be processed by JD Edwards EnterpriseOne. XTS then transforms the response back to the request originator XML format.

Z event

A service that uses interface table functionality to capture JD Edwards EnterpriseOne transactions and provide notification to third-party software, end users, and other JD Edwards EnterpriseOne systems that have requested to be notified when certain transactions occur.

Z table

A working table where non-JD Edwards EnterpriseOne information can be stored and then processed into JD Edwards EnterpriseOne. Z tables also can be used to retrieve JD Edwards EnterpriseOne data. Z tables are also known as interface tables.

Z transaction

Third-party data that is properly formatted in interface tables for updating to the JD Edwards EnterpriseOne database.

Index

D

E

F

G

I

P

R

S

Z