JD Edwards EnterpriseOne Tools

Interoperability Guide

9.2

JD Edwards EnterpriseOne Tools
Interoperability Guide

9.2

Part Number: E53546-09

Contents

Preface

Welcome to the JD Edwards EnterpriseOne documentation.

Documentation Accessibility

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

Access to Oracle Support

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

Related Information

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

`http://learnjde.com`

Conventions

The following text conventions are used in this document:

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

1 Introduction to JD Edwards EnterpriseOne Tools Interoperability

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.

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 9.2.

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

   See the JD Edwards EnterpriseOne Applications Installation Guide for your platform and database. ***http://docs.oracle.com/cd/E61420_01/index.htm***

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

2 Understanding Interoperability

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.

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 simplifies 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.

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.

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.

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* .

JD Edwards EnterpriseOne Interoperability

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

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 Call (Out)**	**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

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 Call (Out)**	**Generate XML Output (Out)**	**Flat Files (Out)**	**Batch (Out)**
Connectors	Dynamic Java Connector	Y	CO, Trans, List*	N	N	RTE XAPI Z Events	N	Y	N	N
Connectors	COM Connector	Y	CO, Trans, List*	N	N	RTE XAPI Z Events	N	Y	N	N
EOne Messaging Adapters	Adapter for MQ WebSphere	Y	CO, Trans*	Y	N	RTE XAPI Z Events	N	Y	N	Y
EOne Messaging Adapters	Adapter for MSMQ	Y	CO, Trans*	Y	N	RTE XAPI Z Events	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
Batch Interfaces	OSA (UBE)	N	N	N	N	N	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 is discussed in detail in this document.										

See *XML CallObject*.

See *XML Transaction*.

See *XML List*.

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.

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 .

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.

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 APIs and Business Functions Guide .

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.

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.

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.

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.

You can use JMS Queue or JMS Topic to send Event notifications as web services.

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.

Business Services Server

Business services enable JD Edwards EnterpriseOne to use web services to exchange information with external systems. JD Edwards EnterpriseOne is both 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 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 .*

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.

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

  Java is a portable language, so you can easily tie JD Edwards EnterpriseOne functionality to Java applications. The JD Edwards EnterpriseOne dynamic Java connector supports real-time event processing.
- 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 *"JD Edwards EnterpriseOne Tools Connectors Overview" in the JD Edwards EnterpriseOne Tools Connectors Guide*
.

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*.

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
- Open Data Access

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. When 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

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 Applications Data Interface for Electornic Data Interchange Implementation Guide*

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 Table Conversion Guide* .

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 Report Printing Administration Technologies Guide* .

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
- 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*.

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 WebLogic 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 WebLogic Server or WebSphere	Web Services	Java	Java, Web Services, Database Ops, Business Functions, XML	Java, Web Services, Database Ops, Business Functions, XML

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**
		Application Server				
Connectors	Dynamic Java Connector	All	Point-to-Point	Java	Java APIs	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
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)

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.

3 Understanding Integrations in a SOA Environment

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.

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.

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* .

Event Notification

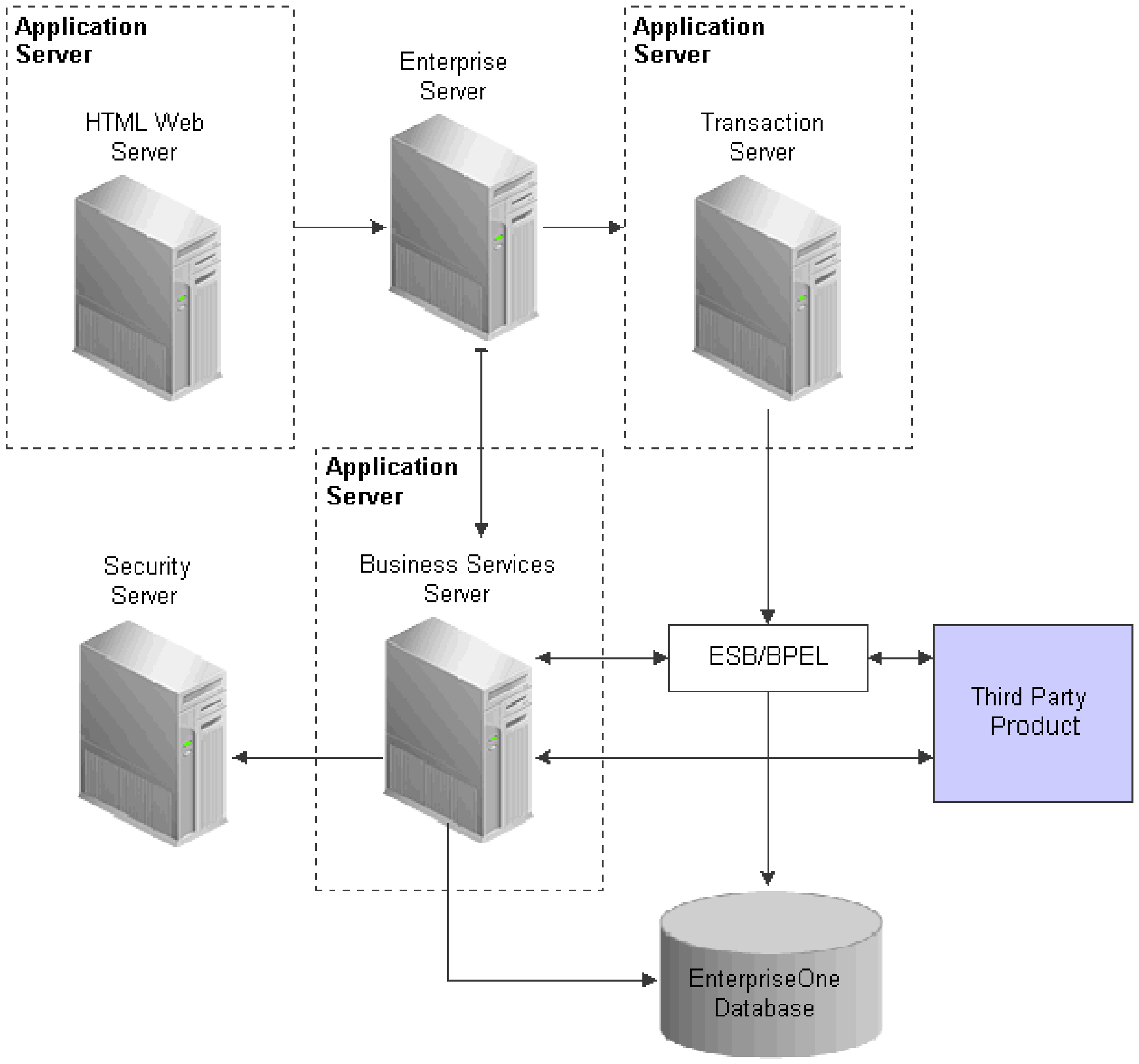
JD Edwards EnterpriseOne sends event notifications as JMS messages through JMS Queue and JMS Topic. The transaction server is the primary business event system for publishing guaranteed event notifications. When a transaction occurs in JD Edwards EnterpriseOne, the transaction server retrieves the data 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.

Note:

- *Understanding Guaranteed Events*.

Business Services Architecture

The following diagram illustrates the 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 WebLogic 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.
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.

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.

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.

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

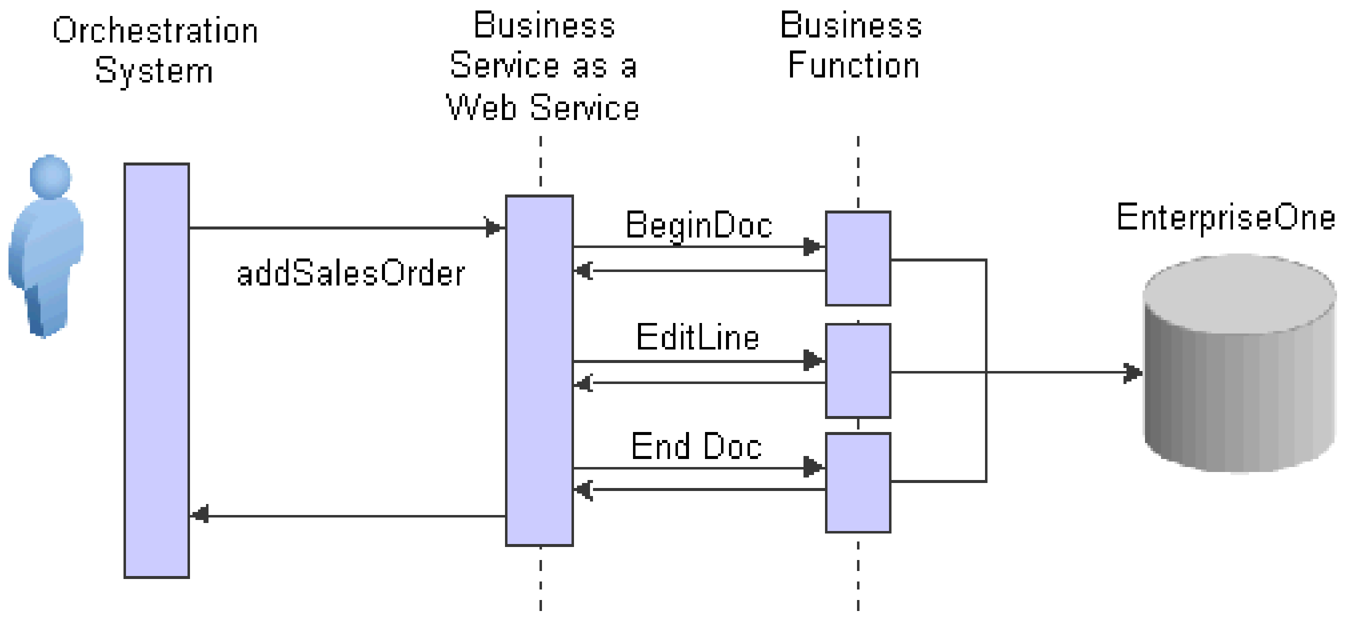
JD Edwards EnterpriseOne supports two methods for processing the web service provider synchronous request/reply pattern. The most frequently used model is to expose a web service that accesses the JD Edwards EnterpriseOne data through a set of business function calls.

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:



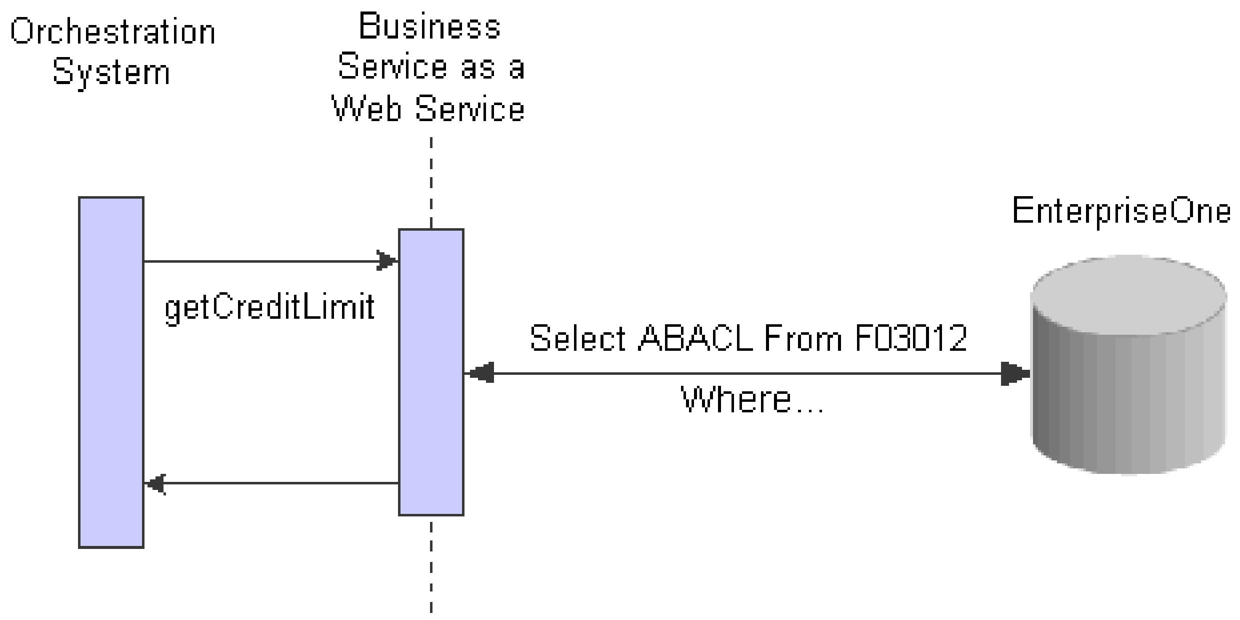
The other method uses JDBj to perform direct data access to the JD Edwards EnterpriseOne database.

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:


Orchestration System
Business Service as a Web Service
EnterpriseOne
getCreditLimit
Select ABACL From F03012 Where...

JD Edwards EnterpriseOne as a Web Service Provider - Asynchronous Notification

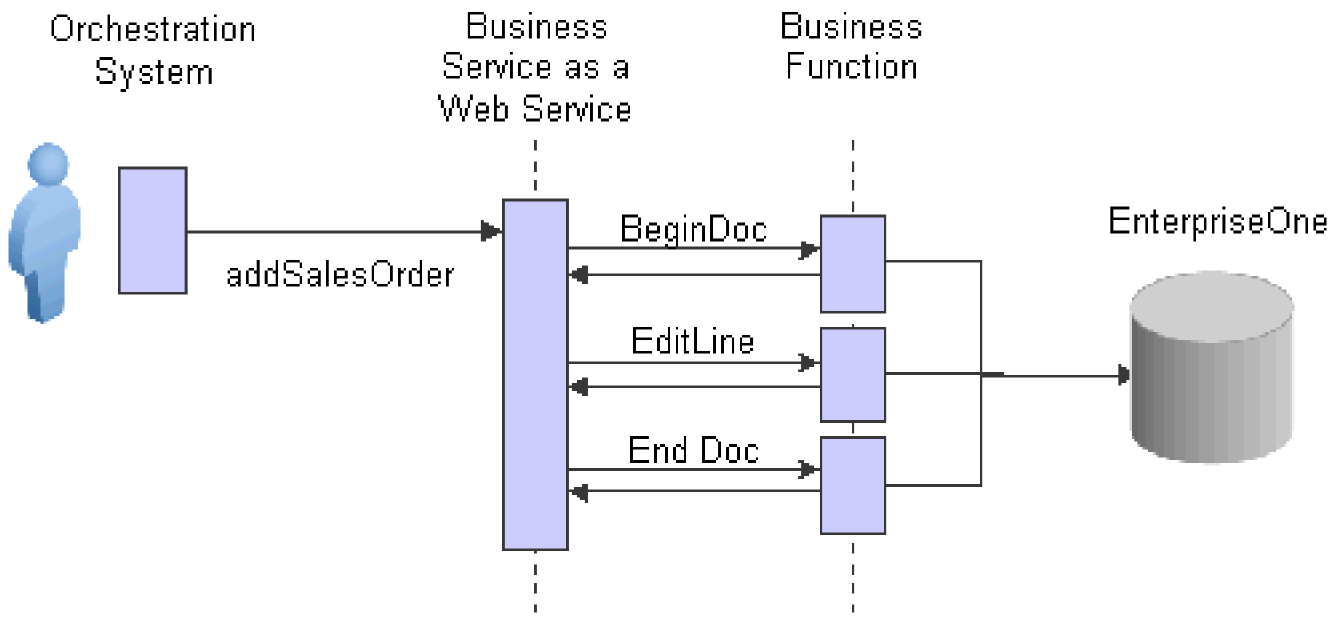
JD Edwards EnterpriseOne supports two methods for processing the web service provider asynchronous notification pattern. The most frequently used method is to expose a web service that accesses the JD Edwards EnterpriseOne data through a set of business function calls.

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:



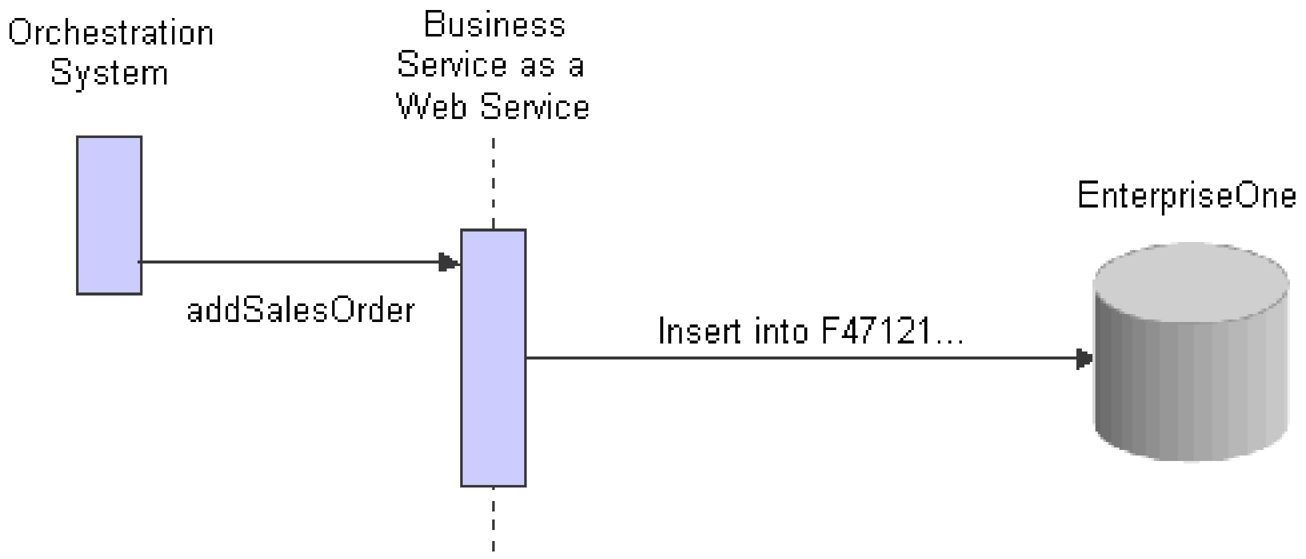
The other method uses JDBj to perform direct data access to the JD Edwards EnterpriseOne database.

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:



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

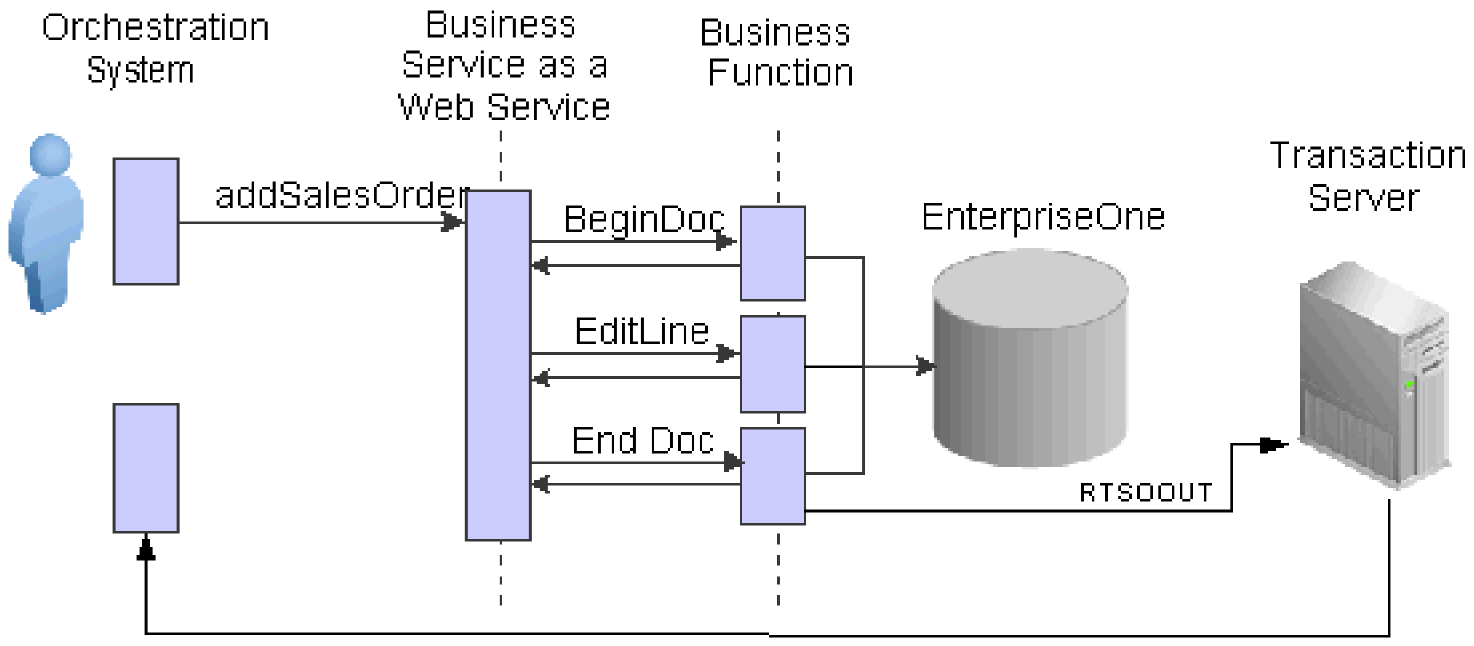
JD Edwards EnterpriseOne supports the web service provider asynchronous request/reply pattern. This method exposes a web service that accesses the JD Edwards EnterpriseOne data through a set of business function calls.

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:



JD Edwards EnterpriseOne as a Web Service Consumer - Notification

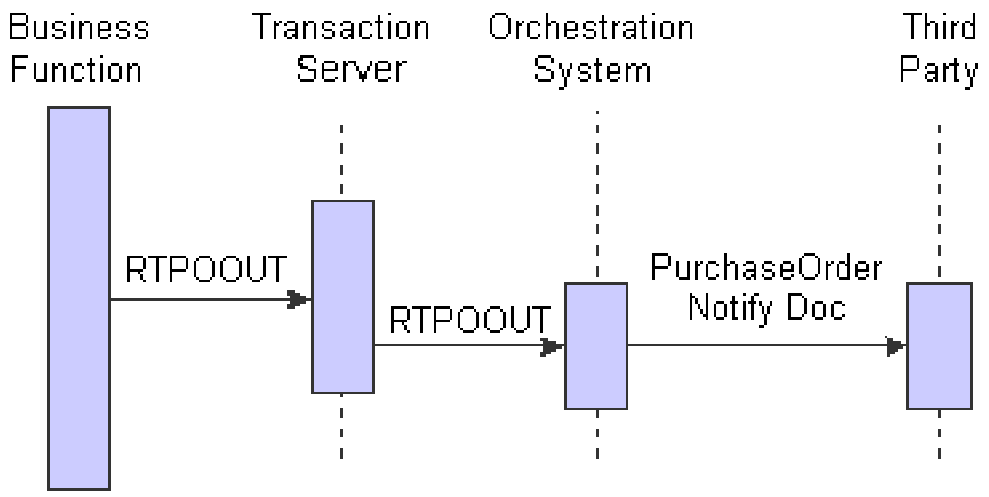
JD Edwards EnterpriseOne supports two methods for processing the web service consumer asynchronous notification pattern. The most frequently used method is to publish a real-time event using the transaction server.

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:



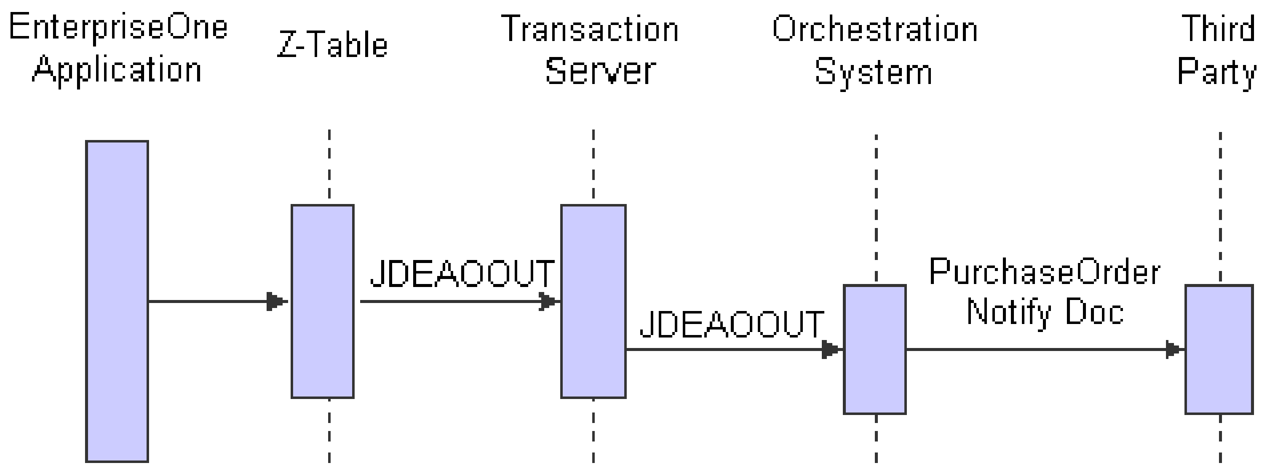
The other method uses Z-tables to send information to third-party systems.

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:



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

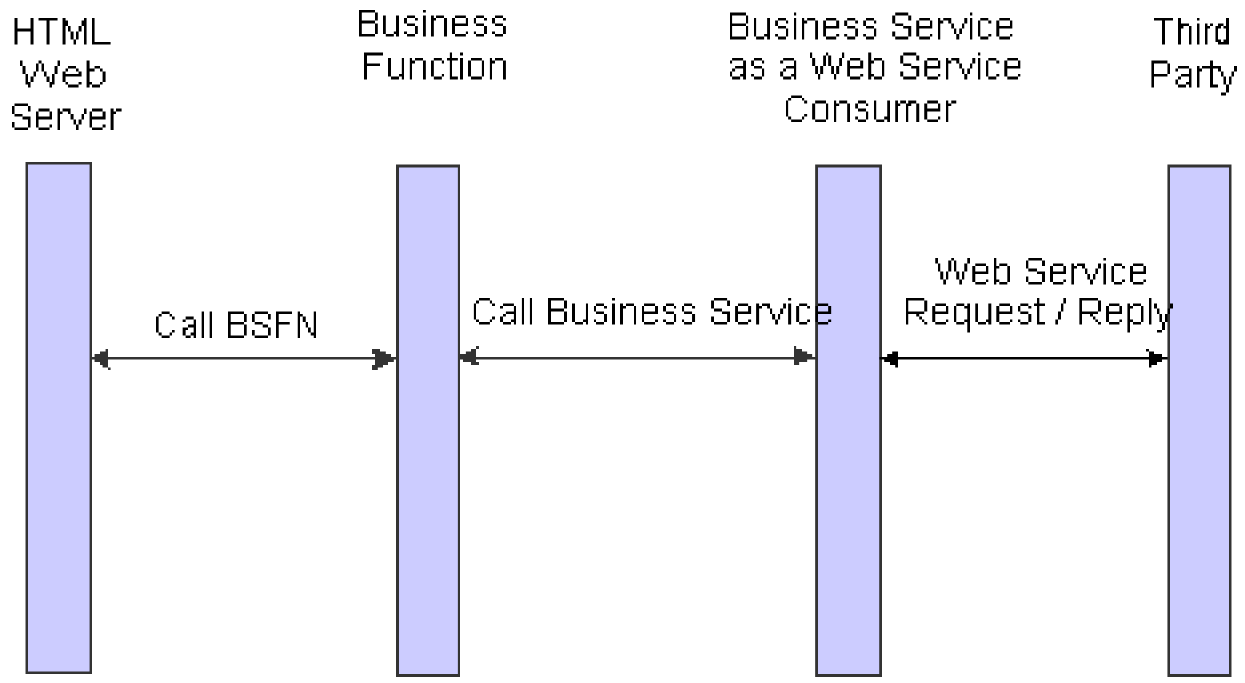
JD Edwards EnterpriseOne supports using a web service for processing the synchronous request/reply pattern. This method uses a JD Edwards business service to call an external web service.

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:



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

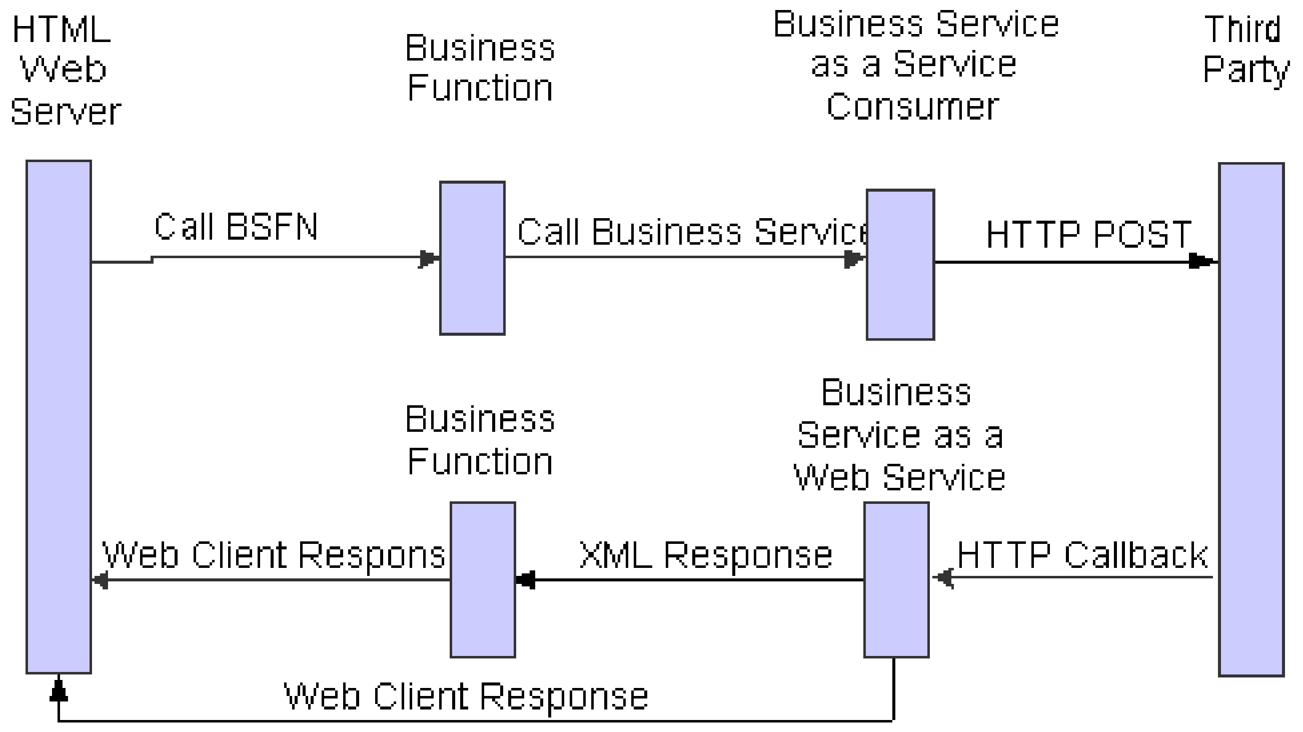
JD Edwards EnterpriseOne supports using HTTP POST for processing an asynchronous HTTP request/response pattern. This method uses HTTP POST as the request and expects an HTTP callback. In this pattern, the web server client continues to process other information while waiting for the response.

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:



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

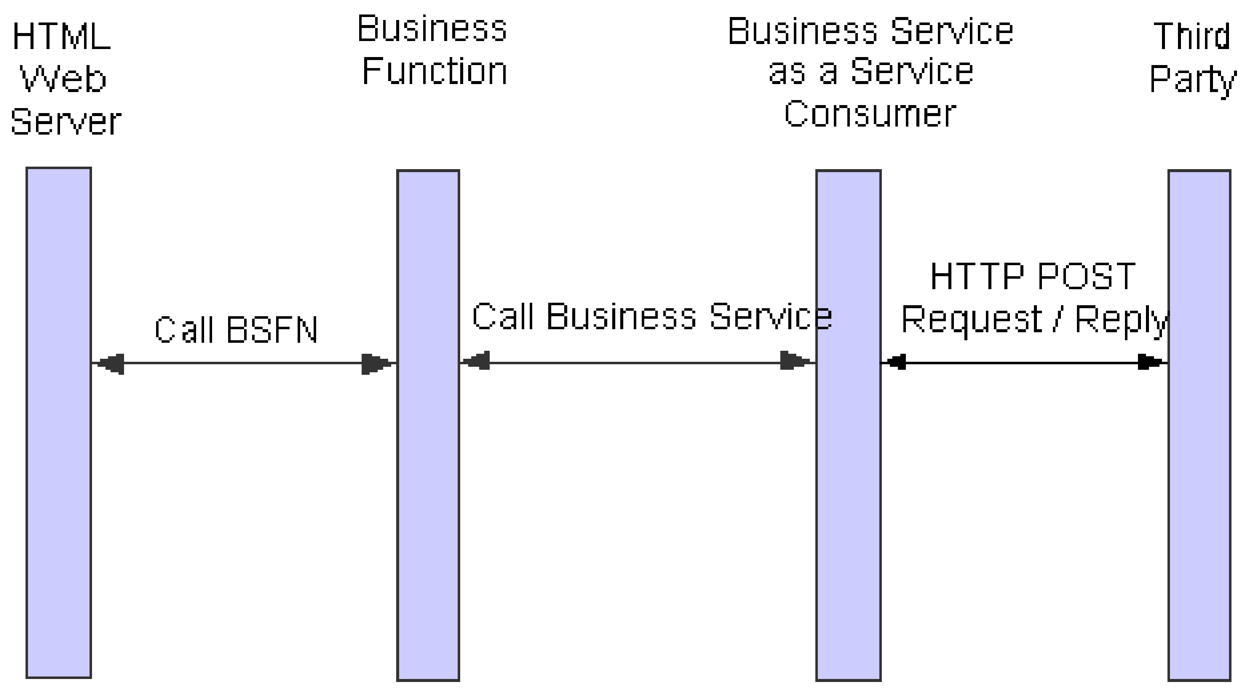
JD Edwards EnterpriseOne supports using HTTP POST for processing a synchronous HTTP request/response pattern. This method uses HTTP POST as the request and waits for the response from the third-party system.

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:



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

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 APIs and Business Functions Guide .

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.

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 parameters that indicates 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 APIs and Business Functions Guide* .

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

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* .

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 Cross Reference Facility Guide .*

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 Event Rules Guide* .

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

5 Understanding XML

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.
XML Dispatch	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.

XML JAR Files

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

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

- Base_JAR.jar
- commons-logging.jar
- httpclient.jar
- httpcore.jar
- JdeNet_JAR.jar
- jmxremote_optional.jar

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
- Commit/Rollback/End
- Terminate Session

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
- Commit/Rollback/End

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.

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.

Here is an example to establish a session using a token instead of a password.

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

Expire Session

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

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=''>
<startTransaction trans='t1' type='manual' />
</jdeRequest>
```

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 is 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>
```

Commit/Rollback/End

Manual transaction sets can be committed or rolled back. Commit, rollback, and end 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 *commit, rollback,* or *end.*

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>

<?xml version='1.0'?>
<jdeRequest type='callmethod' user='steve' pwd='xyz' environment='prod'
role='*ALL' session=''
<endTransaction trans='t1' action='end'/>
</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.

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>
```

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.

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.

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

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	&	&amp;	&#38;
Left angle bracket (less than)	<	&lt;	&#60;
Right angle bracket (greater than)	>	&gt;	&#62;
Straight quotation mark	"	&quot;	&#34;
Apostrophe	'	&apos;	&#39;
Percent	%	Not Applicable	&#37;

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.

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, icudt521.dat, is generally located in system/locale/xml. Use the appropriate setting for your platform.

With JD Edwards EnterpriseOne Tools Release 9.2, the Java Runtime Engine (JRE) is no longer bundled into the tools code. Therefore, before installing EnterpriseOne Tools 9.2, access the vendor website to download and install a 32-bit JRE for your platform.

- For Microsoft Windows, Linux, and Solaris the Java vendor is Oracle.
- For AIX and IBM i the Java vendor is IBM.
- For HP-UX the Java vendor is HP.

Note: Ensure that a certified Java Run Engine (JRE) has been downloaded from Oracle Technology Network (OTN) and is available at the same location as EnterpriseOne. Customers must conform to the supported platforms for the release, which can be found in the Certifications tab on My Oracle Support: `https://support.oracle.com`. Search for product = JD Edwards EnterpriseOne Enterprise Server.

This section discusses:

- UNIX/Linux
- IBM i
- Microsoft Windows

UNIX/Linux

For UNIX/Linux systems, the ICU_DATA path is based on the ICU_DATA environment variable. The UNIX/Linux JD Edwards EnterpriseOne user login script sets the ICU_DATA environment variable to the directory location of the ICU resource file, icudt521.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.

AIX

Verify these environment variable configurations for an AIX platform:

```
export LD_LIBRARY_PATH=<JAVA_HOME>/jre/lib:<JAVA_HOME>/jre/bin/classic:${LD_LIBRARY_PATH}
export LIBPATH=<JAVA_HOME>/jre/lib:<JAVA_HOME>/jre/bin/classic:${LIBPATH}
export SHLIB_PATH=<JAVA_HOME>/jre/lib:<JAVA_HOME>/jre/bin/classic:${SHLIB_PATH}
```

Solaris

Verify these environment variable configurations for a Solaris platform:

```
export LD_LIBRARY_PATH=<JAVA_HOME>/jre/lib/sparc/server:<JAVA_HOME>/jre/lib/sparc:
${LD_LIBRARY_PATH}
export SHLIB_PATH=<JAVA_HOME>/jre/lib/sparc/server:<JAVA_HOME>/jre/lib/sparc:${SHLIB_PATH}
```

Note: Ensure 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.

Linux

Verify these environment variable configurations for a Linux platform:

```
export LD_LIBRARY_PATH=<JAVA_HOME>/jre/lib/i386/server:<JAVA_HOME>/jre/lib/i386:
${LD_LIBRARY_PATH}
export SHLIB_PATH=<JAVA_HOME>/jre/lib/i386/server:<JAVA_HOME>/jre/lib/i386:${SHLIB_PATH}
```

HP-UX

Verify these environment variable configurations for an HP-UX platform:

```
export LD_LIBRARY_PATH=<JAVA_HOME>/jre/lib/IA64N/server:<JAVA_HOME>/jre/lib/IA64N:
${LD_LIBRARY_PATH}
export SHLIB_PATH=<JAVA_HOME>/jre/lib/IA64N/server:<JAVA_HOME>/jre/lib/IA64N:${SHLIB_PATH}
```

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.

Microsoft Windows

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

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.

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

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.

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.

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.

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.

XML Dispatch jde.ini File Configuration

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

[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?

Platform	dispatchDLLName	dispatchDLLFunction
Solaris, Linux, AIX, or HP-UX	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.

[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 before giving up on the response.

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.

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 RUNUBEXML command; however, this command works only on the JD Edwards EnterpriseOne Enterprise server.

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 .*

7 Understanding XML Transformation Service

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.

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

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, and XML list) can process XML documents that are in native format only. 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.

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>
```

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>
```

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.

XTS APIs

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

- IXTSMTemplateSelector
- IXTSMTemplateSelector.TemplateFetchException

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);
  }
 }
}
```

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.

[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
Solaris, Linux, AIX, or HP-UX	libxtskrnl.so	JDEK_DispatchXTS

Other jde.ini File settings include:

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

[JDENET]

Configure this setting:

```
maxKernelRanges=24
```

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

[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=

[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

XML CallObject

XML CallObject is XML-based interoperability that runs as a JD Edwards EnterpriseOne kernel process. You can 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.

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.

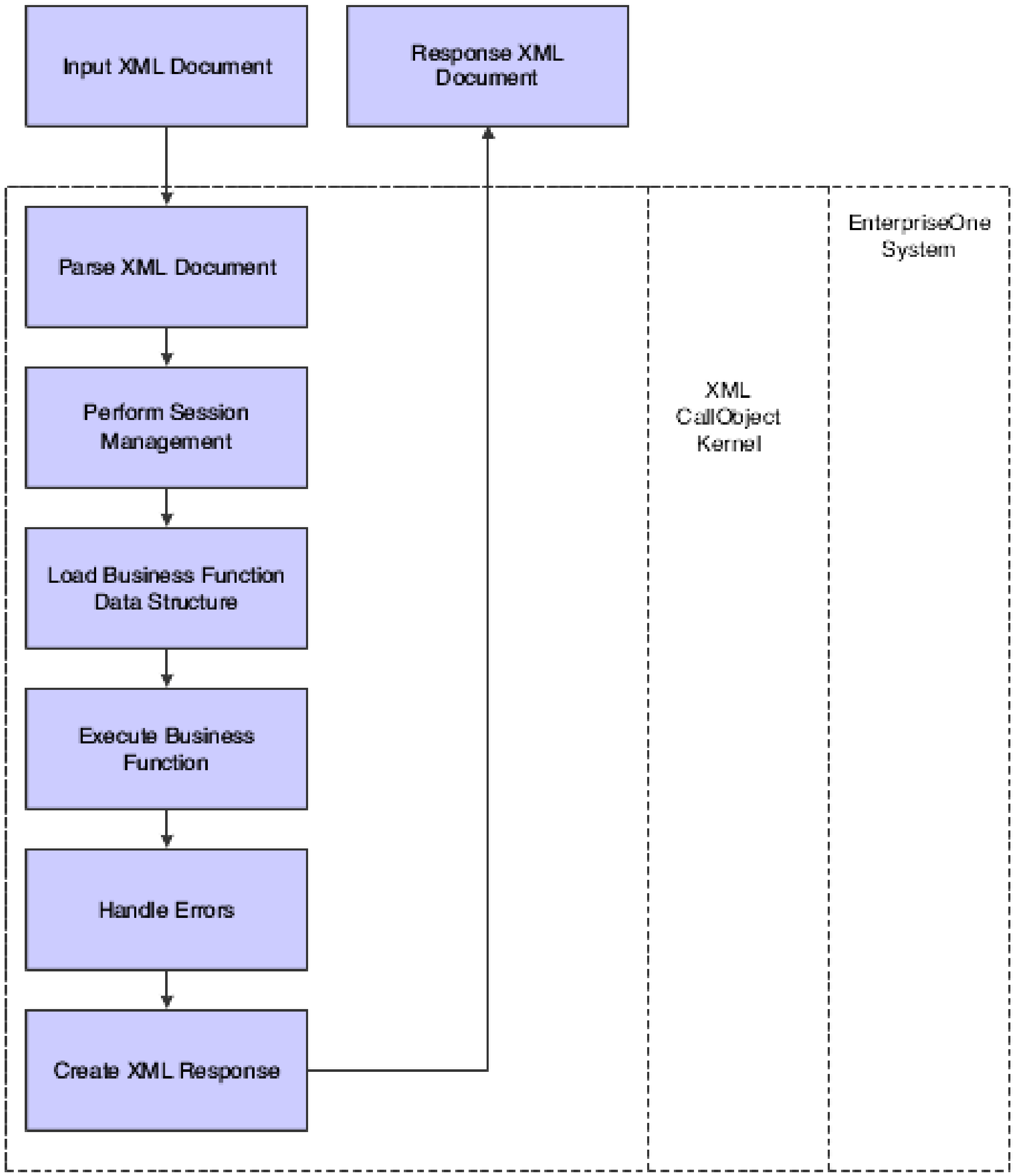
```
<?xml version='1.0' ?>
<jdeRequest type='callmethod' user='' pwd='' environment='prod' role='*ALL'
session=''>
<callMethod name='myfunc' app='P42101'>
<params>
<param name='CostCtr'></param>
<param name='ExpDate'></param>
<param name='Quantity'></param>
</params>
</callMethod>
```

```
</jdeRequest>
```

See *XML Format Examples (All Parameters)*.

XML CallObject Process

This diagram illustrates XML CallObject processing:



In summary:

- The JD Edwards EnterpriseOne Server receives an XML document from the interoperability client.
- 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.

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

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

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 required to give only 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.

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>
```

Note: If you are using XML CallObject with a standard named event rule (NER), you might not be able to use OnError handling. OnError handling considers errors based on the element **returnCode = Return Value** and depends on the jdeCallObject return value containing the error. A standard NER might use the system function **Set NER Error** to capture and return errors, but OnError handling does not have access to read errors set through the **Set NER Error** element.

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>
```

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>
```

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>
```

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.

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>
```

XML CallObject jde.ini File Configuration

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

[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
Solaris, Linux, AIX, or HP-UX	libxmlcallobj.so	XMLCallObjectDispatch

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>
```

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>
```

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.

Code	Description
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

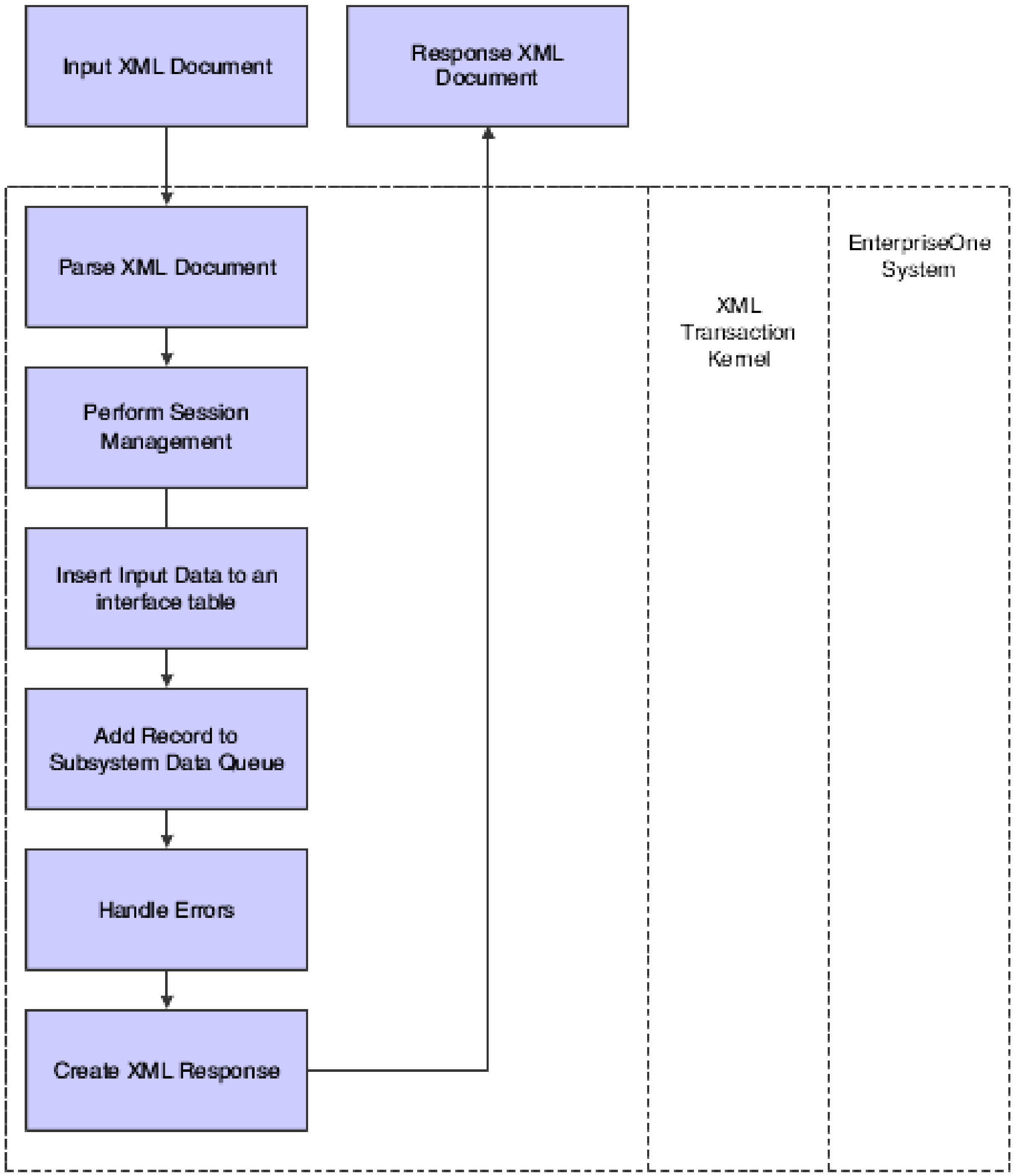
XML Transaction

XML Transaction is XML-based interoperability that runs as a JD Edwards EnterpriseOne kernel process. You 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.

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.



In summary:

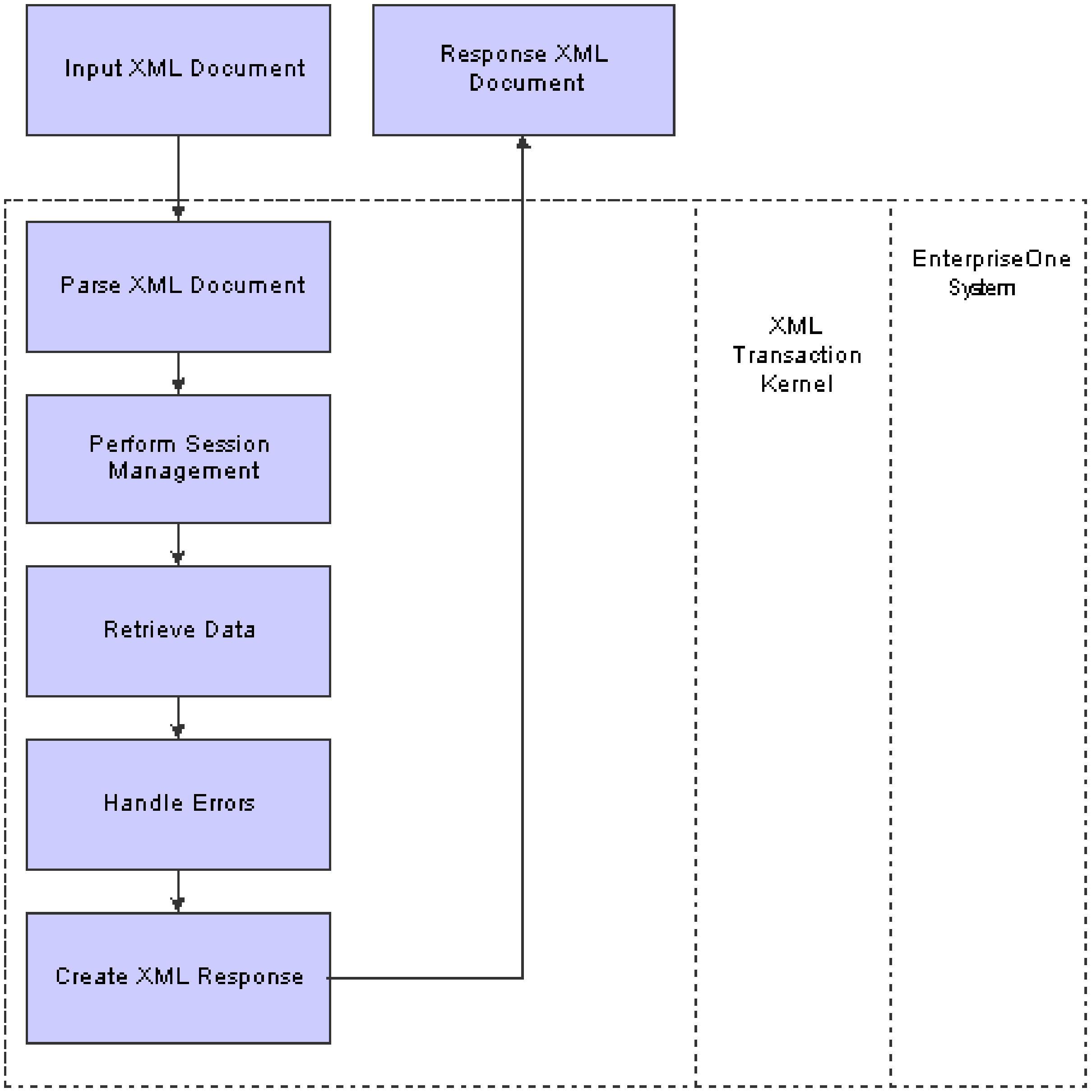
- A request in the form of an XML document contains a list of the data for a predefined transaction type.
- XML Transaction parses the XML inbound document and inserts the data into a JD Edwards EnterpriseOne inbound interface table.
- XML Transaction adds a subsystem data queue record to inform the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

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:



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.

XML Transaction jde.ini File Configuration

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

[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
Solaris, Linux, AIX, or HP-UX	libxmltransactions.so	XMLTransactionDispatch

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 outbound 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>
```

Example: Inbound XML Transaction Request and Response Sample Code

The XML transaction update request is created by the inbound function and sent to the XML transaction API. Thee following code samples illustrate an inbound purchase order request and response. A detailed code sample is provided in Appendix B, XML Format Examples (All Parameters).

See *Inbound XML Transaction Request and Response Format*

This example shows an XML Transaction update request for an inbound purchase order:

```
<?xml version="1.0" encoding="utf-8" ?>
- <!--  This an Inbound Purchase Order.
  -->
- <jdeRequest pwd="password" role="*ALL" type="trans" user="user"
 environment="environment">
- <transaction type="JDEPOIN" action="inbound">
- <key>
  <column name="EdiUserId">TEST</column>
  <column name="EdiTransactNumber">1995598</column>
  <column name="EdiBatchNumber">11004</column>
  <column name="EdiLineNumber">1.000</column>
</key>
</transaction>
</jdeRequest>
```

This example shows the inbound XML Transaction Update response:

```
 <?xml version="1.0" encoding="UTF-8" ?>
- <jdeResponse role="*ALL" type="trans" user="user" xmlns="urn:Schemas-jdedwards-
com:trans.response.JDEPOIN" environment="environment">
- <transaction type="JDEPOIN" action="inbound">
  <returnCode code="0">XML Request OK</returnCode>
- <key>
  <column name="EdiUserId">TEST</column>
  <column name="EdiTransactNumber">2995598</column>
  <column name="EdiBatchNumber">11004</column>
  <column name="EdiLineNumber">1.000</column>
  </key>
  <writeSubsystemRecord>SUCCESS</writeSubsystemRecord>
  </transaction>
  </jdeResponse>
```

10 Understanding XML List

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.

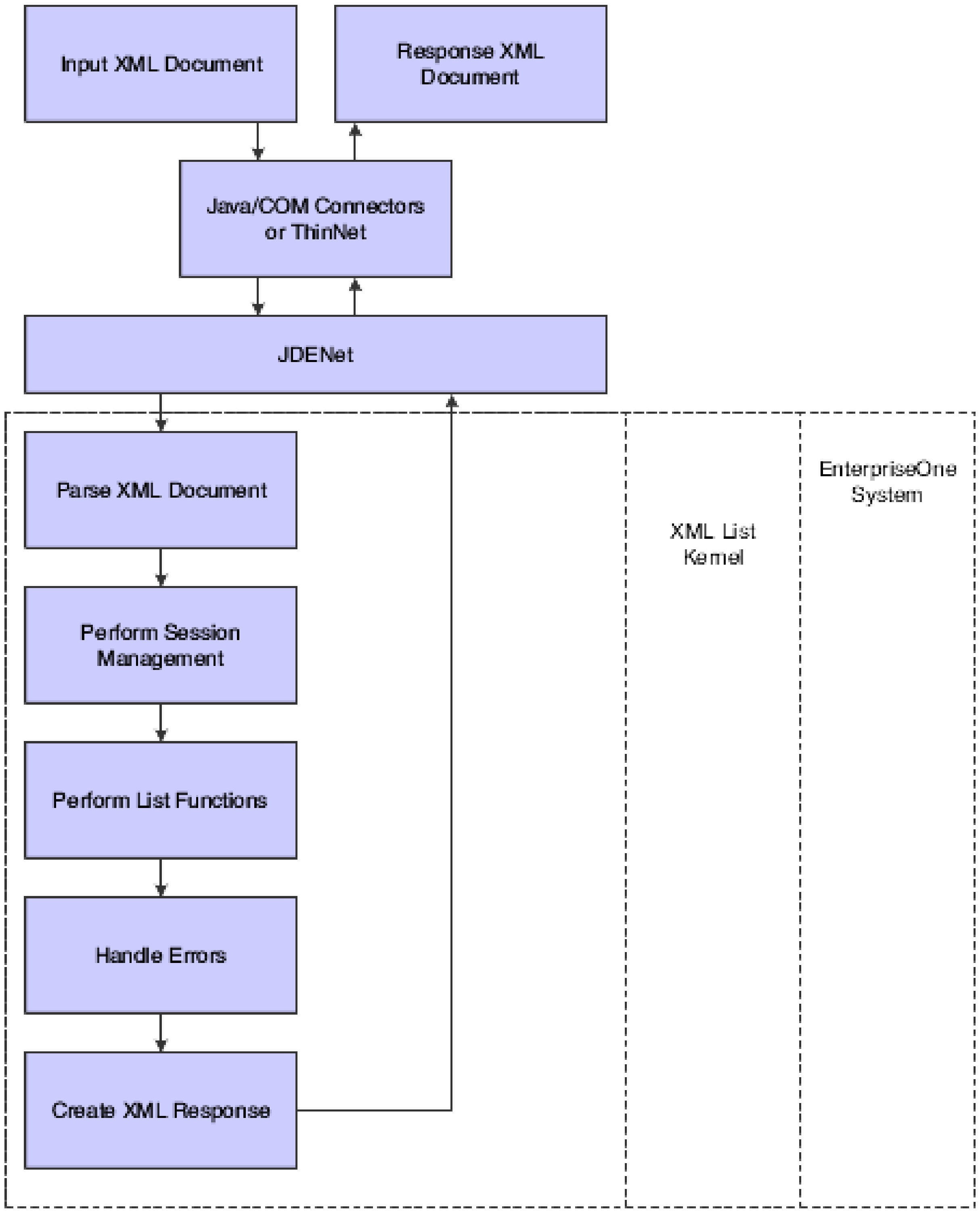
XML List provides a list-retrieval engine that enables you to create an XML data file in the system repository and then retrieve the data in small chunks.

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.

XML List Process

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



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.

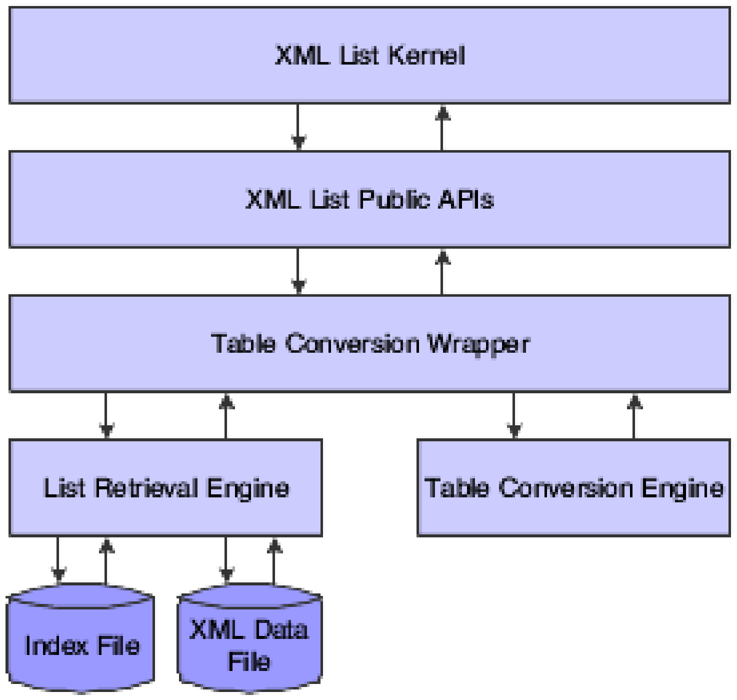
- If the request is a trivial request, XML List retrieves the data and creates a response message to send to the requestor.
- If the request is a non-trivial request, XML List kernel passes the request to the appropriate API:
  - GetTemplate
  - CreateList
  - GetGroup
  - DeleteList
- A table conversion 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.

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:



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:

```
<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.

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>
```

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>
```

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>
```

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.

XML List jde.ini File Configuration

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

[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
Solaris, Linux, AIX, or HP-UX	libxmllist.so	XMLListDispatch

11 Processing Z Transactions

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).

Note:

- *Interoperability Interface Table Information*.

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.

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 APIs and Business Functions Guide* .
- Use Electronic Data Interchange (EDI) to update the interface table.

See *JD Edwards EnterpriseOne Applications Data Interface for Electornic Data Interchange 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.

  > **Note:** 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.

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.

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.

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.

Note: 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.

Note:

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

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.

Note:

- *Using the Revision Application*.

Confirming the Update

This step is optional. If you use a business function, you can create a confirmation function to alert you that a transaction you sent into the JD Edwards EnterpriseOne system has 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.

Task	Explanation

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.

Note:

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

12 Using Flat Files

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.

Some JD Edwards EnterpriseOne batch interfaces, such as the batch extraction programs, can accept flat files and parse the information to data format.

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

Note:

- *JD Edwards EnterpriseOne Interface Tables*.
- *Interoperability Interface Table Information*.
- *"Setting Up Table Conversions" in the JD Edwards EnterpriseOne Tools Table Conversion Guide* .
- *JD Edwards EnterpriseOne Applications Data Interface for Electornic Data Interchange Implementation Guide*

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"

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 table 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 comma (,). 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.

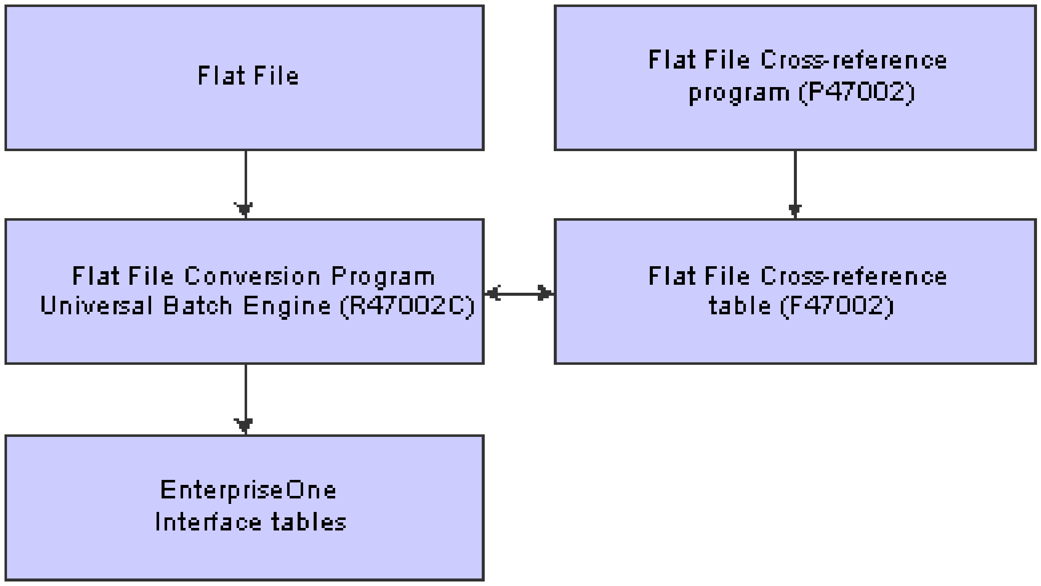
Converting Flat Files Using the Flat File Conversion Program

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:


Flat File
Flat File Cross-reference program (P47002)
Flat File Conversion Program Universal Batch Engine (R47002C)
Flat File Cross-reference table (F47002)
EnterpriseOne Interface tables

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 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.

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.

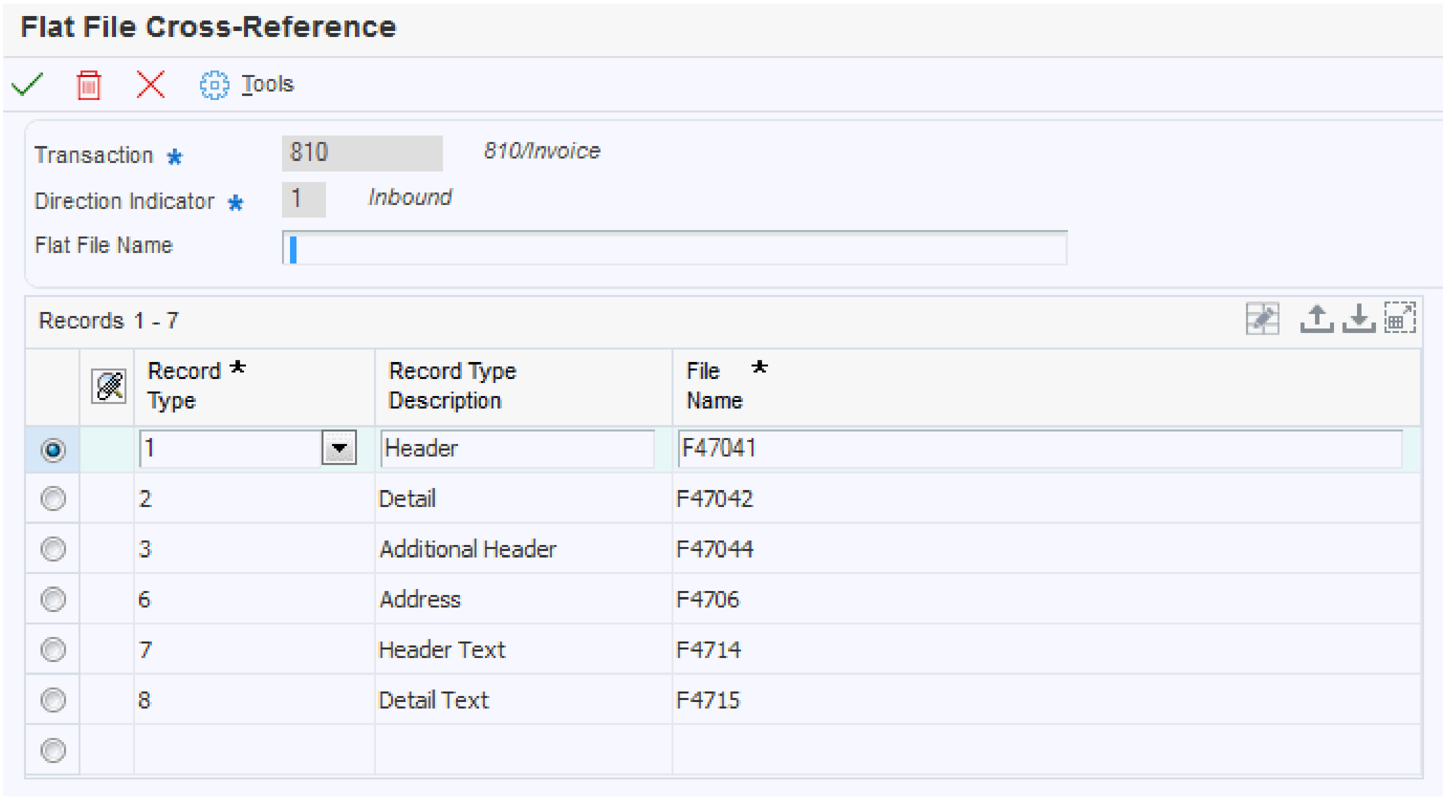
Note:

In the following example, the Sales Order Management application is used as an example. The navigation to the Work With Flat File Cross-Reference for Sales Order Management is:

From the JD Edwards EnterpriseOne Navigator, select EnterpriseOne Menus > Customer Relationship Management > Sales Order Management > Sales Order Advanced and Technical Ops > Sales Interoperability > Flat File Cross-Reference.

Defining the Flat File Cross Reference Table

Access the Flat File Cross Reference form.



Transaction *	810	810/Invoice
Direction Indicator *	1	Inbound
Flat File Name		

Records 1 - 7

	Record Type *	Record Type Description	File Name *
●	1	Header	F47041
○	2	Detail	F47042
○	3	Additional Header	F47044
○	6	Address	F4706
○	7	Header Text	F4714
○	8	Detail Text	F4715
○			

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.

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 supports the business function only 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.

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.

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.

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

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.

Note:

- *"Understanding Foreign Tables" in the JD Edwards EnterpriseOne Tools Table Conversion Guide* .

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.

Setting Up Flat File Encoding

Access the Unicode Flat File Encoding Configuration form from the Windows client and complete the following fields:

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

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.

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*.

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*.

Inbound 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*.

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 *JD Edwards EnterpriseOne Applications Business Interface Reference Guide* provides information for creating real-time 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 Overview" in the JD Edwards EnterpriseOne Applications Business Interface Reference Guide*.

Z Event Processing

A typical flow for processing outbound 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 notification 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.

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 system, 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.

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.

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.

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.

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

Queue	MSMQ Platform and Recommended Name	*IBM i* Platform and Recommended Name	NT Platform and Recommended Name	UNIX Platform and Recommended Name
Default Response	Not applicable	DEFRES.Q	DEFRES.Q	DEFRES.Q

Note: 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.

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.

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.

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.

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.

Default Response Queue

The default response queue stores the processed messages into JD Edwards EnterpriseOne. These messages may be 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.

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

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 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 provides 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

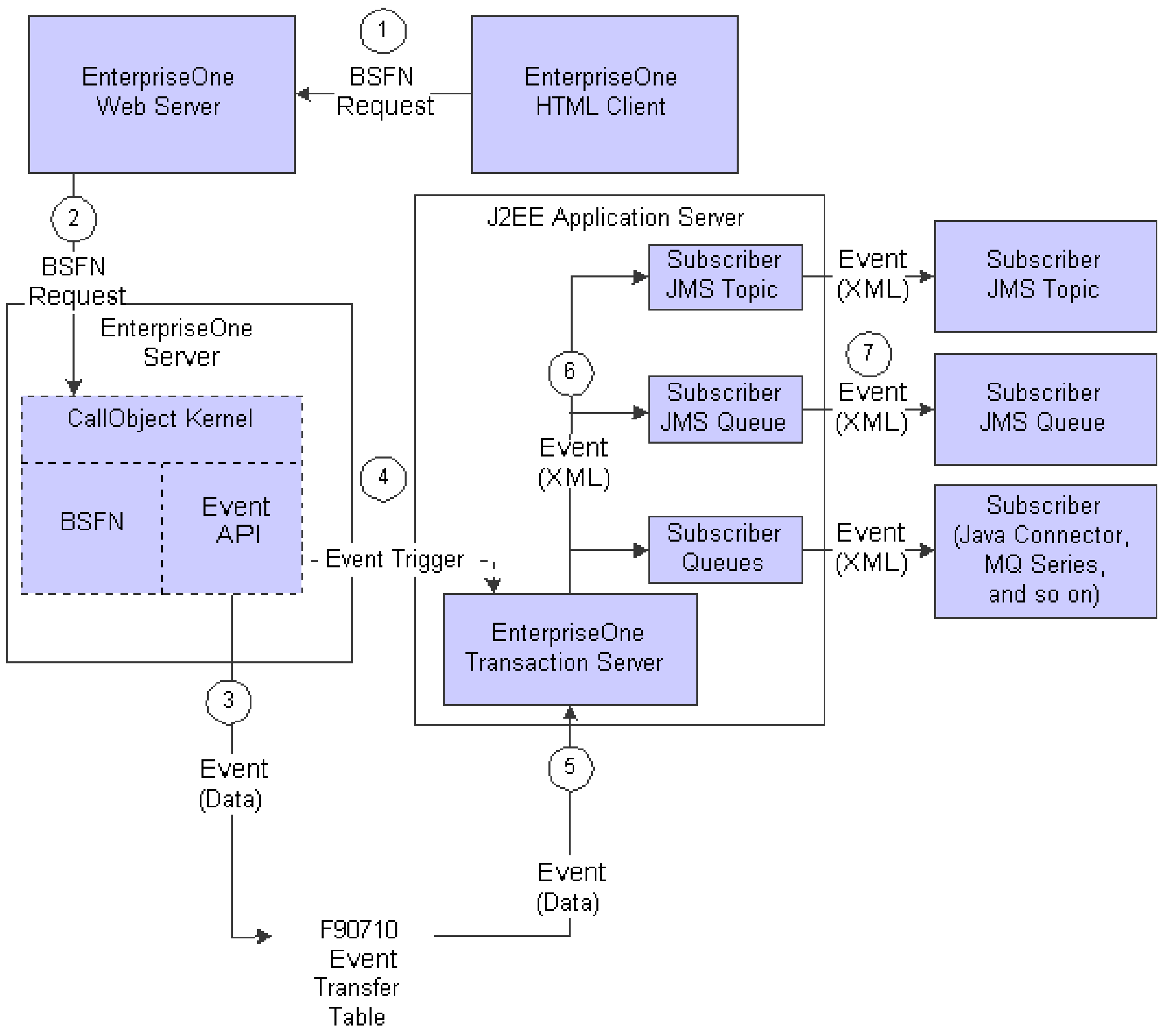
Processing Guaranteed Events

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

- Aggregating events
- Logging events
- Configuring the transaction server

Understanding Guaranteed Events Processing

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



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.

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.

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* .

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.

You configure the Object Configuration Manager (OCM) so that the transaction server can find the event system. You 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.

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.

Understanding OCM Setup for Guaranteed Event Delivery

You 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 *JD Edwards EnterpriseOne Tools System Administration Guide* for information about setting up OCM.

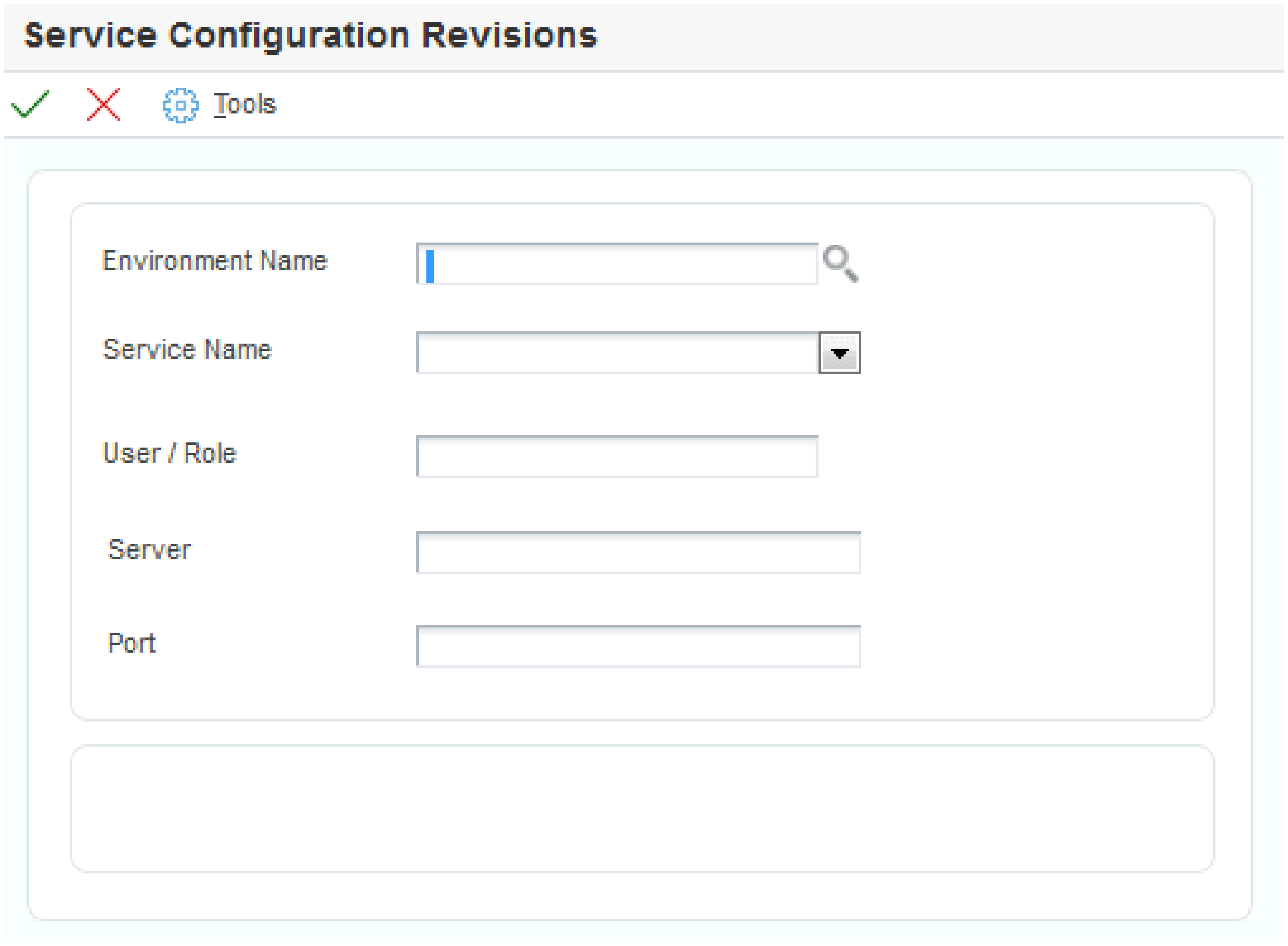
Forms Used to Set Up OCM for Guaranteed Event Delivery

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

Form Name	FormID	Navigation	Usage
			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.

Setting Up the OCM for Guaranteed Event Delivery

Access the Service Configuration Revisions form.



Environment Name	
Service Name	
User / Role	
Server	
Port	

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.

Defining Events

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

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 installation or reference guide.

See the installation or reference guide for your platform and database. *http://docs.oracle.com/cd/E24902_01/index.htm*

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.

Note:
- *Understanding Guaranteed Z Events*.

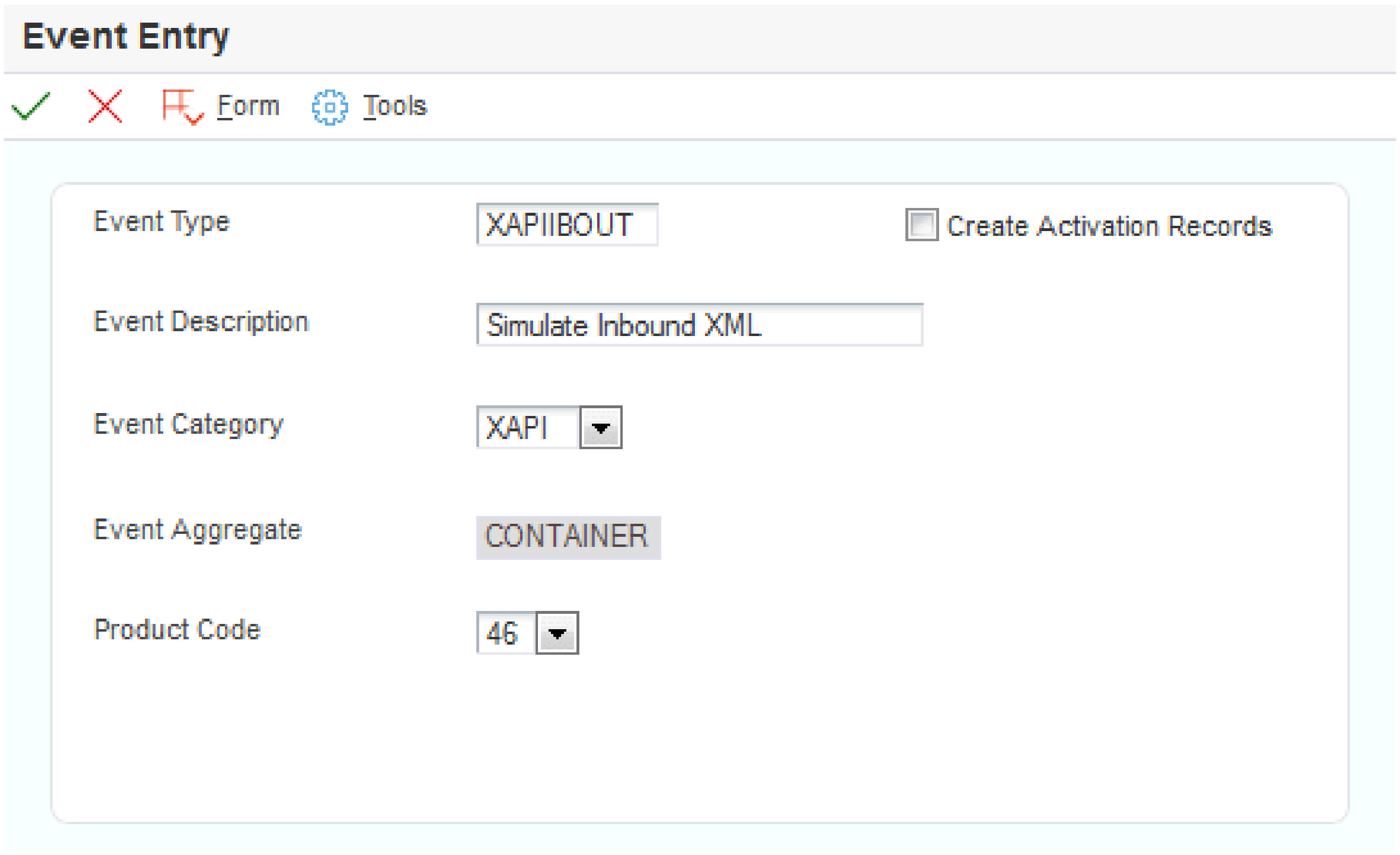
Forms Used to Enter Events

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.

Form Name	FormID	Navigation	Usage
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.

Adding a Single or Container Event

Access the Event Entry form.



Event Type	XAPIIBOUT
Create Activation Records	☐
Event Description	Simulate Inbound XML
Event Category	XAPI
Event Aggregate	CONTAINER
Product Code	46

Event Type
The name of the event (for example RTSOOUT, which is the typical 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.

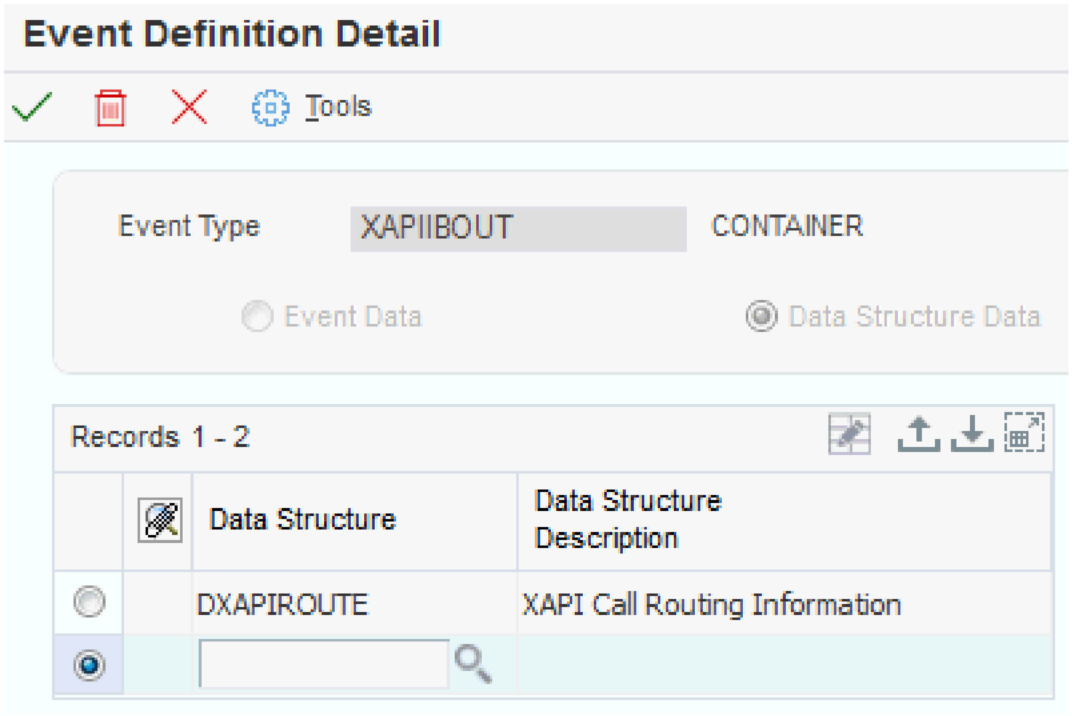
Product Code
An optional field that indicates to which JD Edwards EnterpriseOne system the event is associated.

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.

Event Definition Detail

Access the Event Definition Detail form.


Event Definition Detail

Event Type XAPIIBOUT CONTAINER

Event Data [✓ Data Structure Data]

Records 1 - 2

		Data Structure	Data Structure Description
○		DXAPIROUTE	XAPI Call Routing Information
●			

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.

Activating an Event

Access the Add Event Activation by Environment form

Environment
Your operating environment, such as Microsoft Windows, UNIX, *IBM i* , and so on.

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.

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.

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 Service Bus (OSB) is a subscriber that uses the JMS Queue transport. You can set up processing options for Enterprise Service Bus (ESB) (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.

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.
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.

Form Name	FormID	Navigation	Usage
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.

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.

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

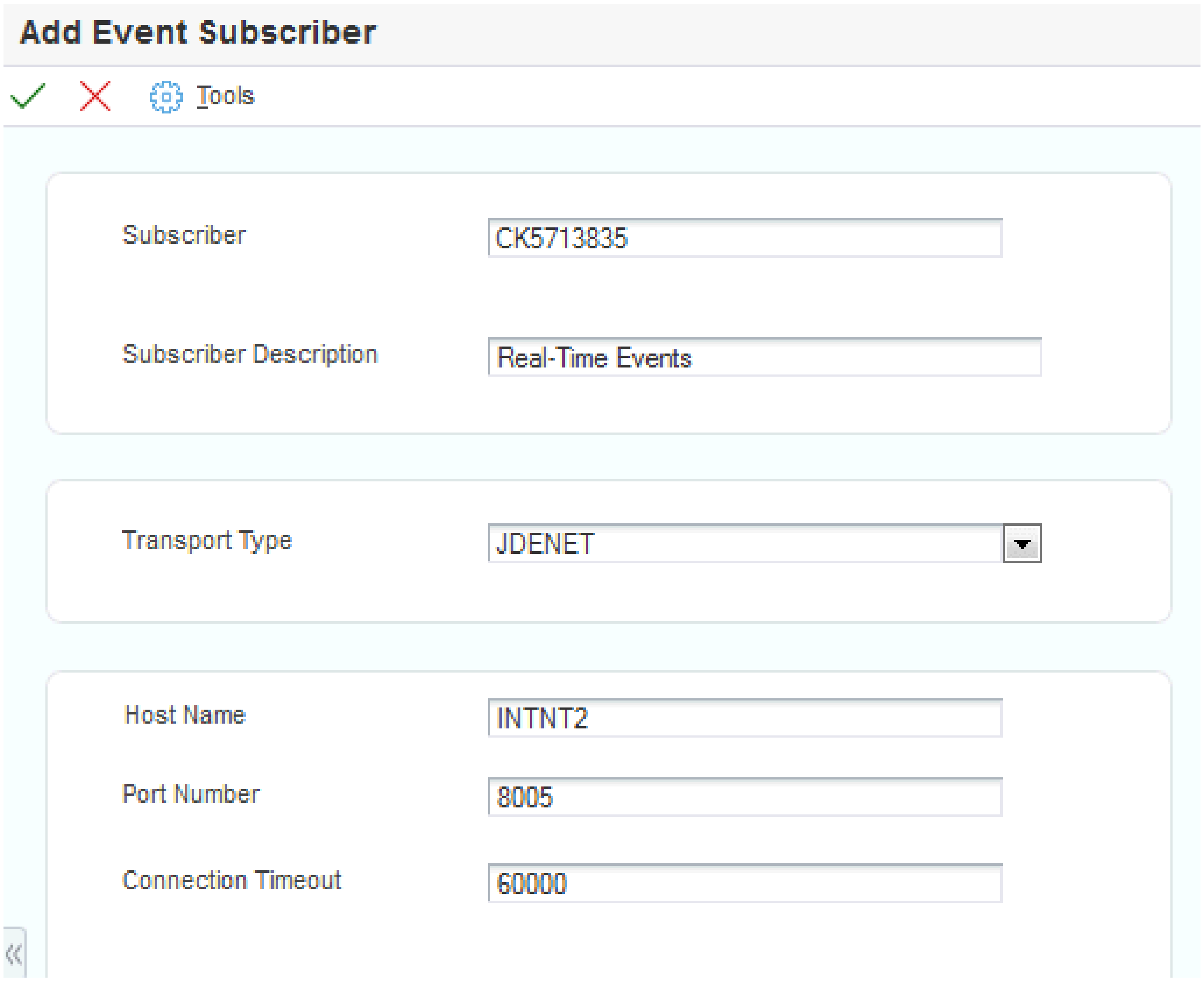
Use this processing option to specify the value for the Provider URL field that appears on the Add Event Subscriber form. The value you enter in this processing option appears in the Add Event Subscriber form when the Application Server is defined as **WebLogic** and the Queue Location is defined as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

Adding a Subscriber

Access the Add Event Subscriber form.



Subscriber	CK5713835
Subscriber Description	Real-Time Events
Transport Type	JDENET
Host Name	INTNT2
Port Number	8005
Connection Timeout	60000

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
- 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 JMS Topic field, enter the JMS Topic name for your subscriber.

  Note: 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 the *JD Edwards EnterpriseOne Transaction Server Reference Guide* for your platform. ***http://docs.oracle.com/cd/E61420_01/index.htm***
- JMSQUEUE: JMS Queue

  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--this 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. Use 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.

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

  See *JD Edwards EnterpriseOne Transaction Server Reference Guide* for your platform. ***http://docs.oracle.com/cd/E61420_01/index.htm***
- 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

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.

Associating a Subscription with Subscribed Events

Access the Subscribed Events form .

Event Type
The name of the event.

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.

Creating MSMQ Queues

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

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

Prerequisites

Install MSMQ on your system.

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.

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.

Verifying Event Delivery

Use these steps to verify event delivery:

1. Start the COMConnector on your enterprise server.

   Note: Do not start the COMConnector on your client machine.

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.

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.

Prerequisites

Before you complete this task:

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

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.

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.

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

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.

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.

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.

Prerequisites

Install the WebLogic server on your system.

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.

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.

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.

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.

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.

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.

Creating Custom Real-Time Events

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

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. This chapter 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 *JD Edwards EnterpriseOne Applications Business Interface Reference Guide* .

See "JD Edwards EnterpriseOne Application Real-Time Events Overview" in the *JD Edwards EnterpriseOne Applications Business Interface Reference 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 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 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* .
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*

   See *JD Edwards EnterpriseOne Deployment Server Reference Guide* for your platform. ***http://docs.oracle.com/cd/E61420_01/index.htm***

   See *JD Edwards EnterpriseOne Transaction Server Reference Guide* for your platform and application server. ***http://docs.oracle.com/cd/E61420_01/index.htm***

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.

Understanding the Schema Generation Utility

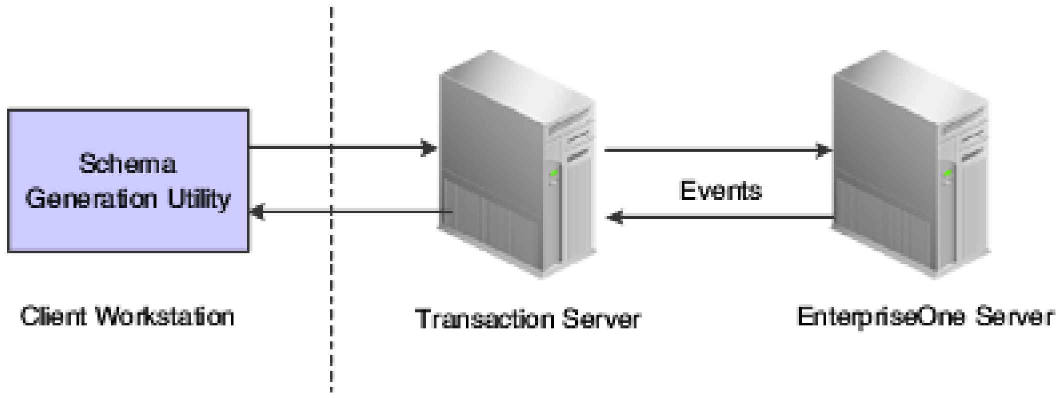
The Schema Generation Utility creates XML schemas from event definitions. The purpose of this utility is to facilitate orchestration developers who use orchestration systems such as Oracle's Enterprise Service BUS (ESB) or Business Process Execution Language Process Manager (BPEL-PM) to process real-time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



Prerequisite

Before you configure the Schema Generation Utility, you must install the certified Java Runtime Environment (JRE) version on your local machine.

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 WebLogic Server, the default value is: weblogic.jndi.WLInitialContextFactory For WebSphere Application Server, the default value is: com.ibm.websphere.naming.WsnInitialContextFactory
[EVENTS]	jndiProviderURL	For the Transaction Server running on a WebLogic Server, the value is: jndiProviderURL=t3://machine_name:machine_port 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.

Section	Setting	Value
[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.

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.

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

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

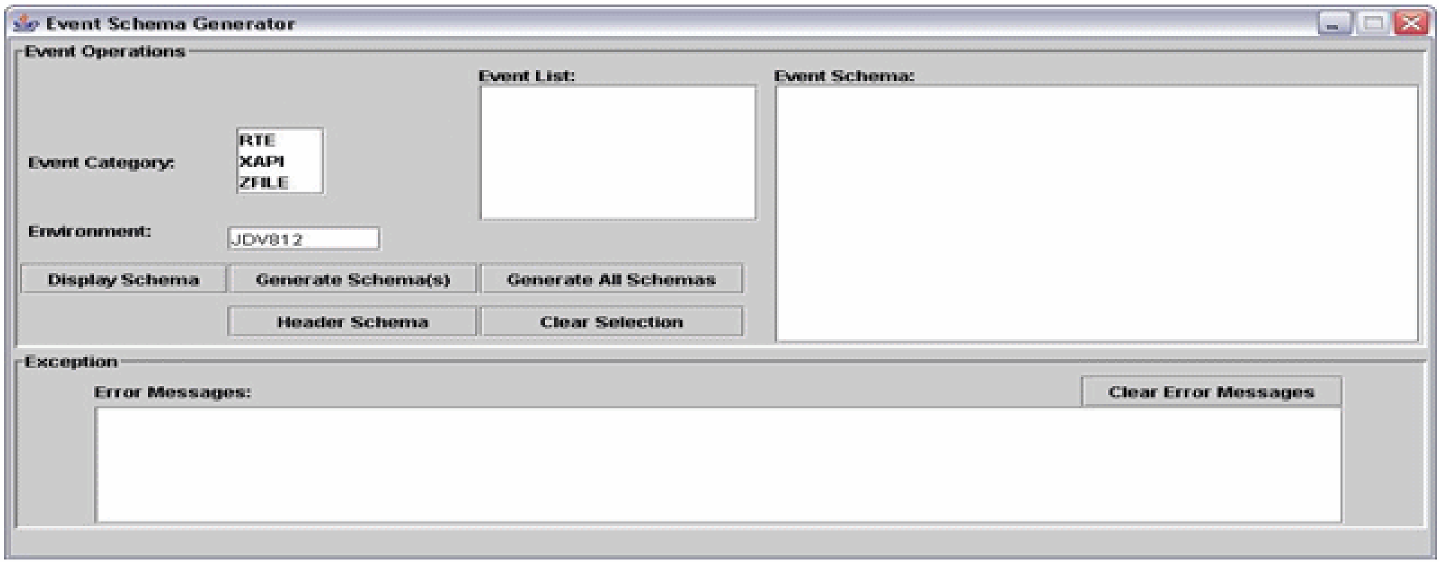
  Events must be active in this environment.
- Role

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

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:



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.

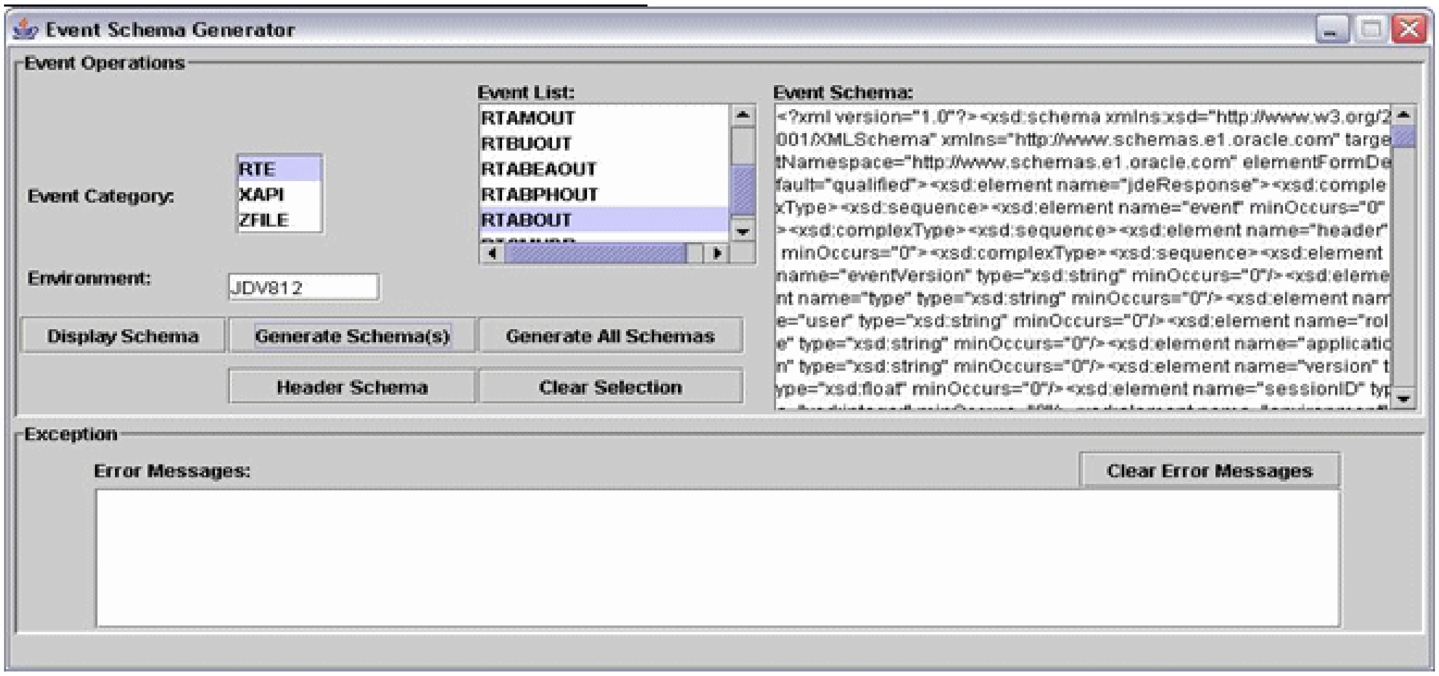
If an error occurs during schema generation, the utility displays an error message in the Error Messages field of the Exception panel. You remove a message by clicking the Clear Error Message button.

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.

Displaying Event Schema

You can display event schema. This example shows how the utility displays a schema:



Environment: JDV812

Event List:
RTAMOUT
RTBUOUT
RTABEAOUT
RTABPHOUT
RTABOUT

Event Schema:

```
<?xml version="1.0"?><xsd:schema xmlns:xsd="http://www.w3.org/2
001/XMLSchema" xmlns="http://www.schemas.e1.oracle.com" targe
tNamespace="http://www.schemas.e1.oracle.com" elementFormDe
fault="qualified"><xsd:element name="jdeResponse"><xsd:comple
xType><xsd:sequence><xsd:element name="event" minOccurs="0"
><xsd:complexType><xsd:sequence><xsd:element name="header"
 minOccurs="0"><xsd:complexType><xsd:sequence><xsd:element
name="eventVersion" type="xsd:string" minOccurs="0"/><xsd:eleme
nt name="type" type="xsd:string" minOccurs="0"/><xsd:element nam
e="user" type="xsd:string" minOccurs="0"/><xsd:element name="rol
e" type="xsd:string" minOccurs="0"/><xsd:element name="applicatio
n" type="xsd:string" minOccurs="0"/><xsd:element name="version" t
ype="xsd:float" minOccurs="0"/><xsd:element name="sessionID" typ
```

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.

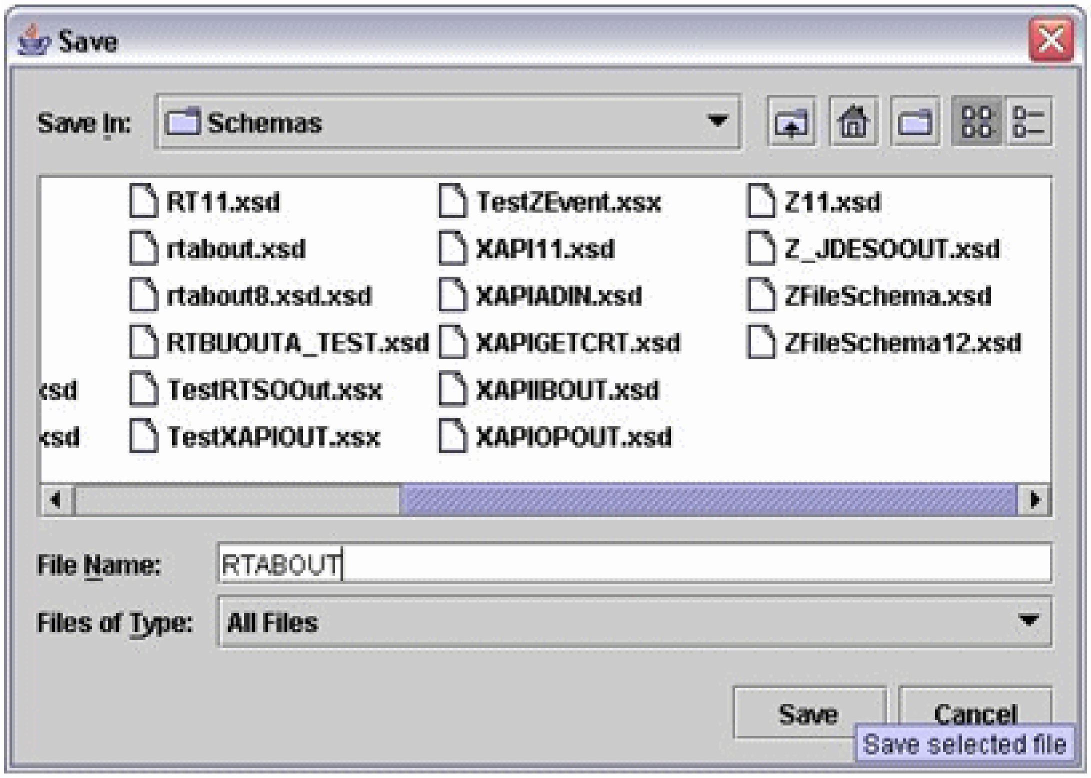
   Select only one event. If you select multiple events, the utility displays an error message in the Error Messages field of the Exception panel. The error message for selecting multiple events indicates invalid input. Click the Clear Selection button to clear a selection from the Event List.
4. Click Display Schema.

The utility displays the generated schema in the Event Schema field.

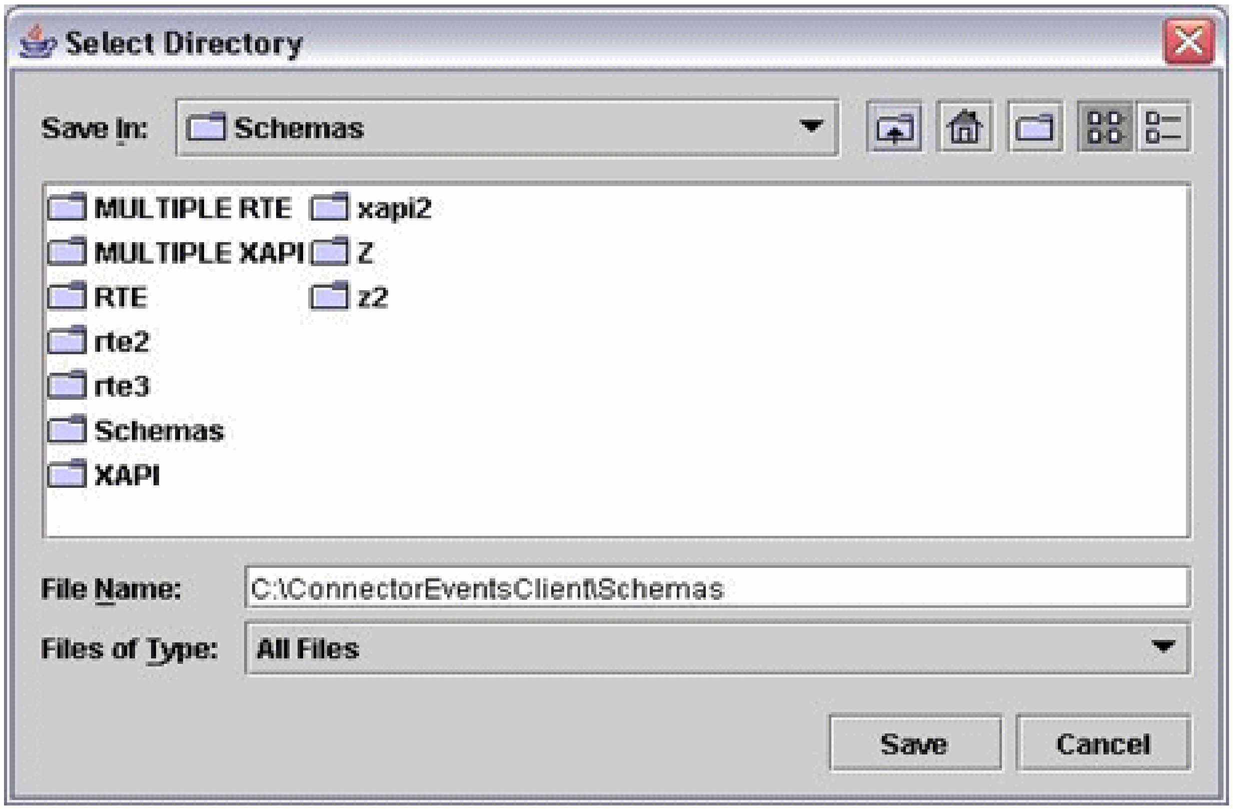
Generating Event Schema for Single and Multiple Events

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:



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.



File Name: C:\ConnectorEventsClient\Schemas

Files of Type: All Files

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).

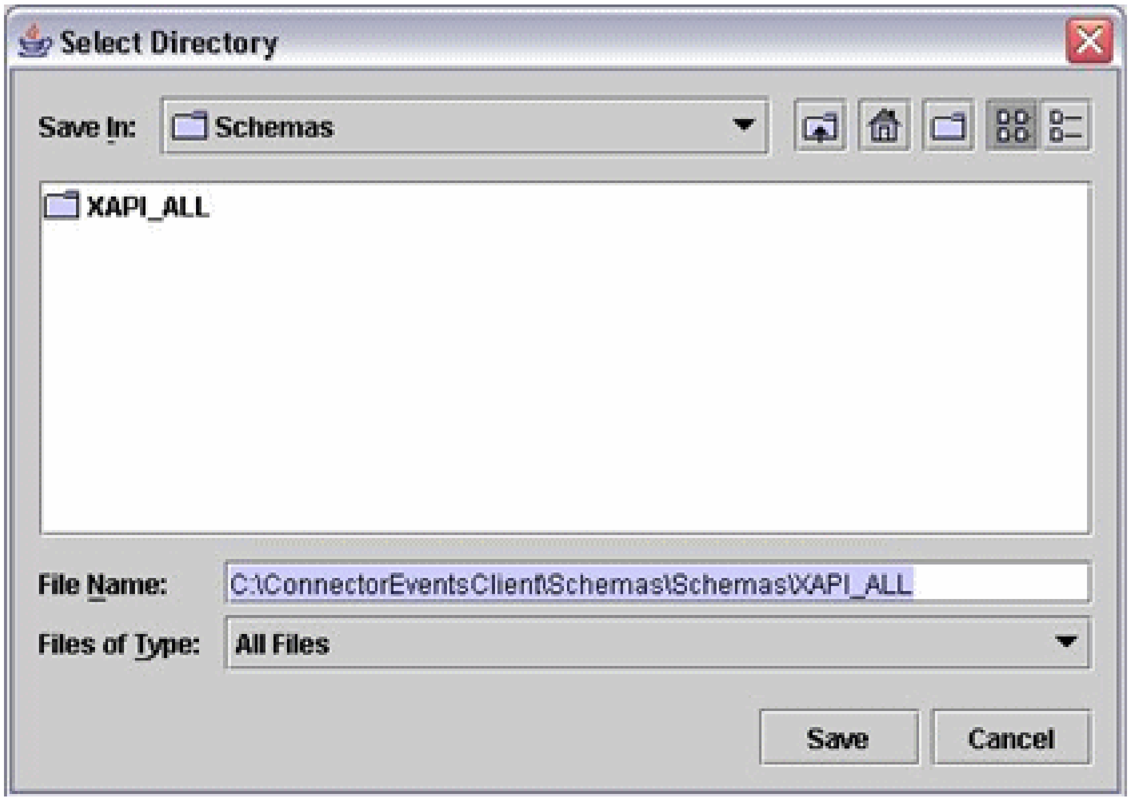
     A file chooser dialog screen appears. To save the generated schema, navigate to the appropriate directory, enter a name for the generated schema in the File Name field, and click Save. If you do not want to save the generated schema, click Cancel.
   - Select two or more events from the Event List field, 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.

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:



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.

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.

Problem	Resolution
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

Versioning Real-Time Events

This section is intended for software development engineers who customize existing JD Edwards EnterpriseOne real-time events.

This section provides an overview about why you would create a version of a real-time event and discusses how to:

- Determine if a version is required
- Name a real-time event version

Understanding Why a Version Is Required

JD Edwards EnterpriseOne ships pre-built real-time events to which you can subscribe. You can use the pre-built real-time events out of the box, you can customize an existing JD Edwards EnterpriseOne real-time event, or you can create a new real-time event that is specific for your enterprise.

Note:

- *"Introduction to JD Edwards EnterpriseOne Business Interfaces" in the JD Edwards EnterpriseOne Applications Business Interface Reference Guide*
- *Creating Custom Real-Time Events* in this chapter.

If you find it necessary to change an existing real-time event and the change affects the interface, Oracle recommends that you create a version of the original real-time event and change the version as necessary. This recommendation takes into consideration JD Edwards EnterpriseOne updates and upgrades. During an update or upgrade, any modifications that you made to an existing JD Edwards EnterpriseOne real-time event will be overwritten by the new JD Edwards EnterpriseOne code. A real-time event version helps to minimize merge complications when your system is upgraded.

Note: The conventions discussed in this section are standards that Oracle JD Edwards EnterpriseOne uses. You can use these standards as a model for creating your own standards so that your versions will not be overwritten.

Determining if a Version is Required

If you make a change to an existing event and that change affects the interface, you need to create a new version of the event.

The interface is affected when you:

- Change the event type.
- Change the event data.

Changing the Event Type

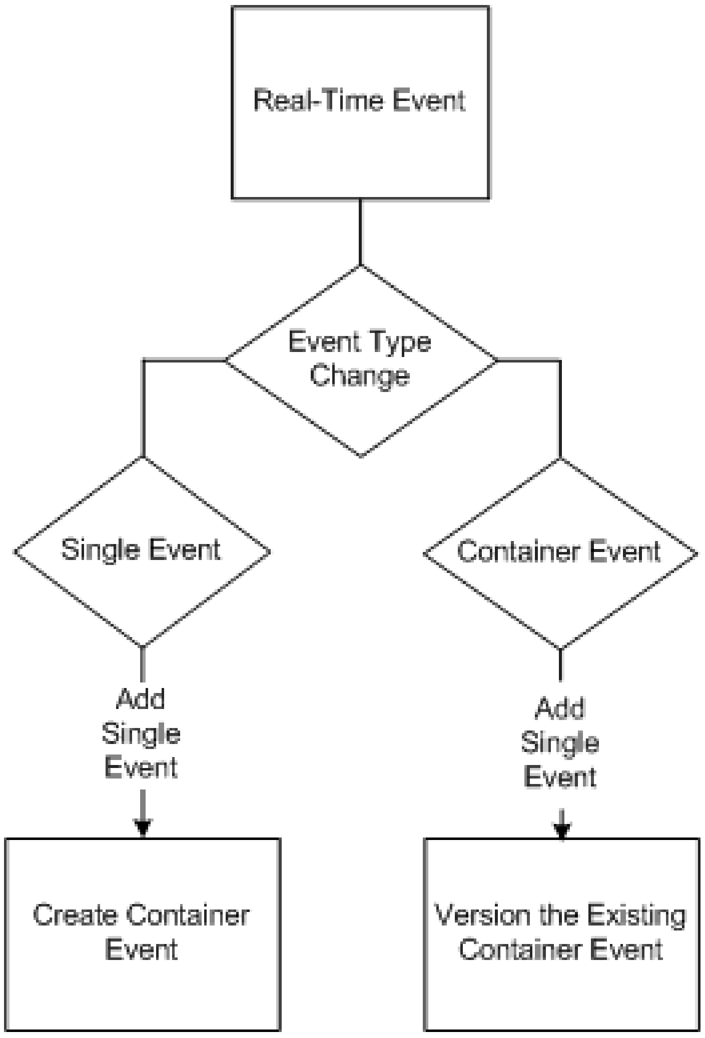
An event can be either a single event or a container event. Each single event consists of a data structure. A container event consists of multiple single events.

When you are working with a single event, and you determine that you need to add another single event that is associated with the existing single event, you must create a new container event that contains both the original single event and the added single event. A container event is named differently than a single event. For example, if the original single event is named RTSOHDR, and you want to add another single event named RTSODTL, you create a new container event called RTSOOUT that contains the two single events, RTSOHDR and RTSODTL.

A container event contains more than one single event, and if it becomes necessary to add another single event, you create a version of the existing container event and add the desired single event.

Oracle recommends that you not remove any single events from the system. If you want to remove a single event from an existing container event, this can be achieved through the subscription process when you associate your subscription with the events to which you want to subscribe. See *Establishing Subscriber and Subscription Information* in this chapter.

The following diagram can help you decide if you should create a version of the original real-time event because you have changed the event type:



Changing the Event Data

A single event uses one data structure to pass information to other applications. Because a container event is composed of multiple single events, the container event uses multiple data structures to pass information to other applications. A data structure is a collection of data items. The event will not behave as expected or you will receive an error if you change the data type or data size fields within the data item. Oracle recommends that you not change the data type or data size.

Changes to the event structure, such as the structure name, data type, and data size affect the interface. If you make a change that affects the event interface, you must create a version of the event and make the necessary changes in the version.

The event interface could change because you:

- Change the data structure member name.
- Change the data type.
- Change the data size.
- Add a new field.
- Remove a field.

- Populate an existing field that previously was not populated.
- Discontinue populating an existing field that previously was populated.
- Change the source of the data; for example, the data is retrieved from a different table.

If you make any of the above changes, you must create a version of the event, and you must make a change to the business function that initiates the event.

Note: Changing when an event is initiated cannot be accommodated by creating a version of an existing event.

Naming Conventions for Real-Time Event Versions

This section discusses the standards that Oracle uses for naming real-time events. If you determine that an existing real-time event should be changed, you should make a copy of the original event and update the copy. The copy becomes a version. When you change an existing real-time event, the name must clearly indicate that it is a version of an original real-time event. This enables users of real-time events to choose the appropriate version.

Oracle uses the following naming convention for real-time events:

RTXXYYY

where XX is the product area (for example, AB, PO, or SO) and YYY is the type (for example, OUT, HDR, DTL). An example event name is RTSOHDR.

When Oracle creates a new version of an existing real-time event, the naming convention is:

RTXXYYYN

Where N is the next sequence number for the version, with the first version starting with 2. For example, the original real-time event is named RTSOHDR. The first time you create a version of an event, the name is RTSOHDR2. The next time you create a version of the same event, the name is RTSOHDR3.

Note: These examples for naming events are standards that Oracle uses. If you modify an existing JD Edwards EnterpriseOne real-time event for your enterprise, ensure that you use a naming convention that will not be overridden when your system is upgraded.

References to Other Documentation

Additional detailed technical information about real-time events is available in the Oracle Technical Catalog, that can be found by clicking the Resources tab on the JD Edwards Upgrade Resources Web page:

http://www.upgradejde.com

For more information about existing real-time events, see the *JD Edwards EnterpriseOne Applications Business Interface Reference Guide* .

15 Using Guaranteed Real-Time Events

Understanding Guaranteed Real-Time Events

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.

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.

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.

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

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 Guaranteed XAPI Events

Understanding Guaranteed XAPI Events

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.

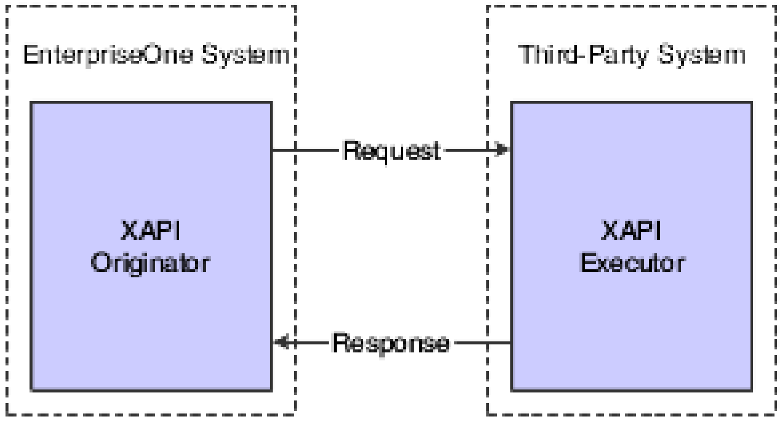
The XAPI structure sends outbound events and receives replies. An event is first generated by the XAPI originator and then sent to a separate system, the XAPI executor, for processing. The XAPI executor then sends a response back to the XAPI originator. The XAPI structure provides for these three possibilities 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.

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:

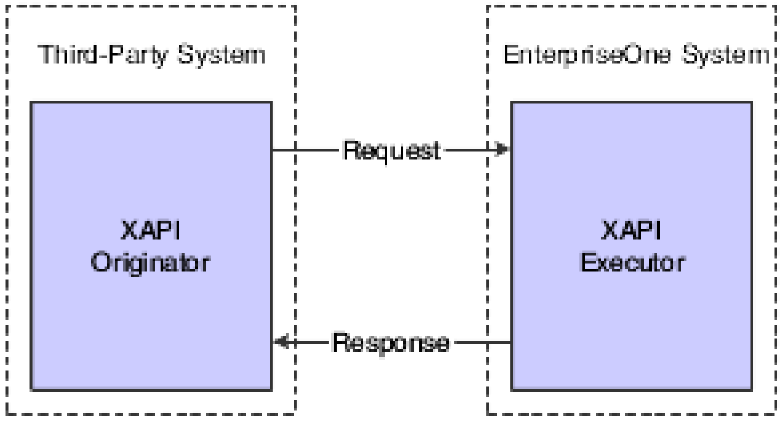


In summary:

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.

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:

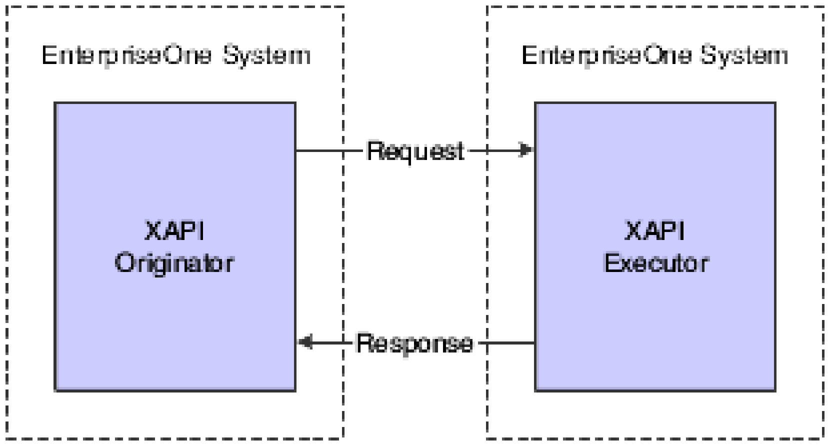


In summary:

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.

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:



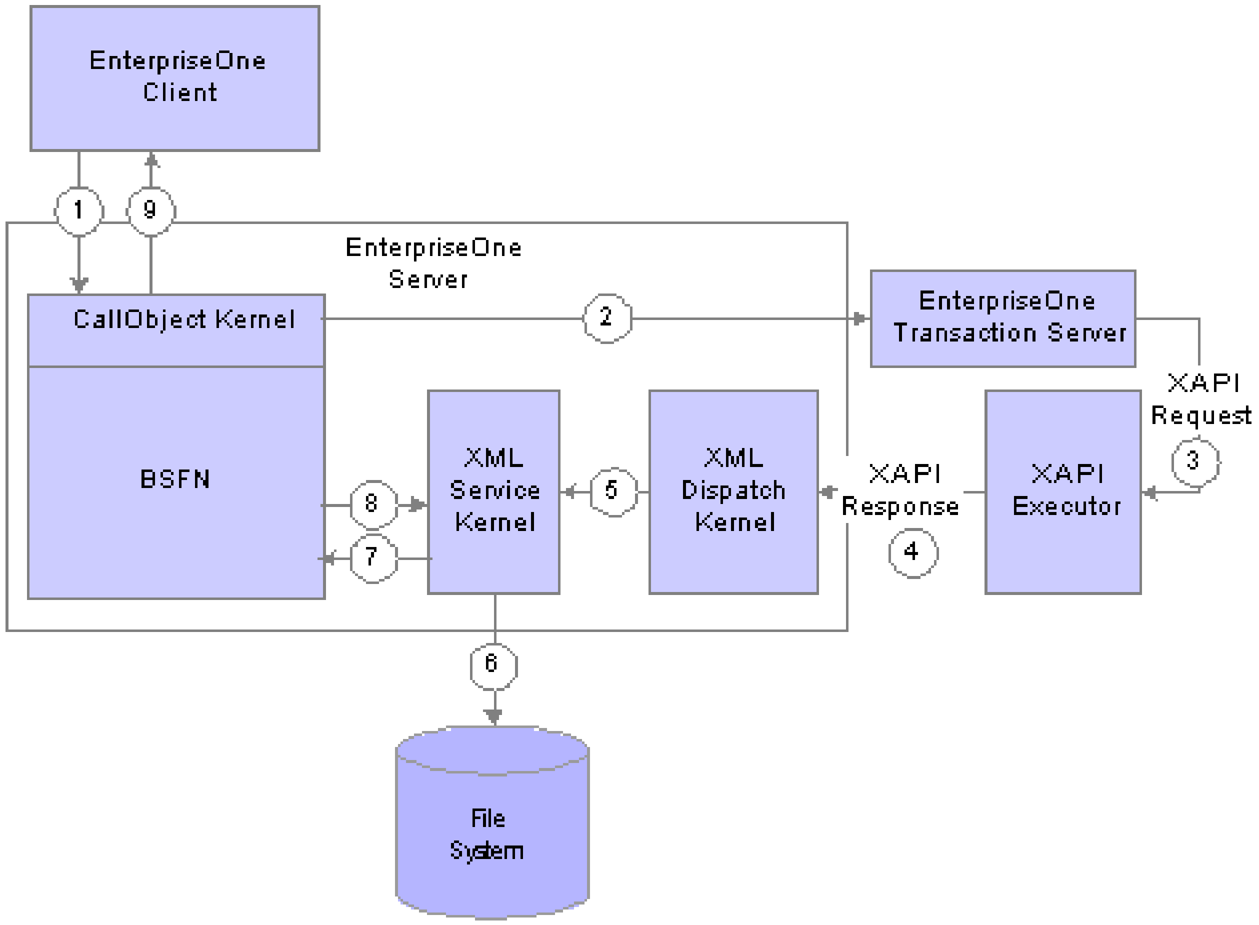
In summary:

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.

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:



In summary:

1. 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.

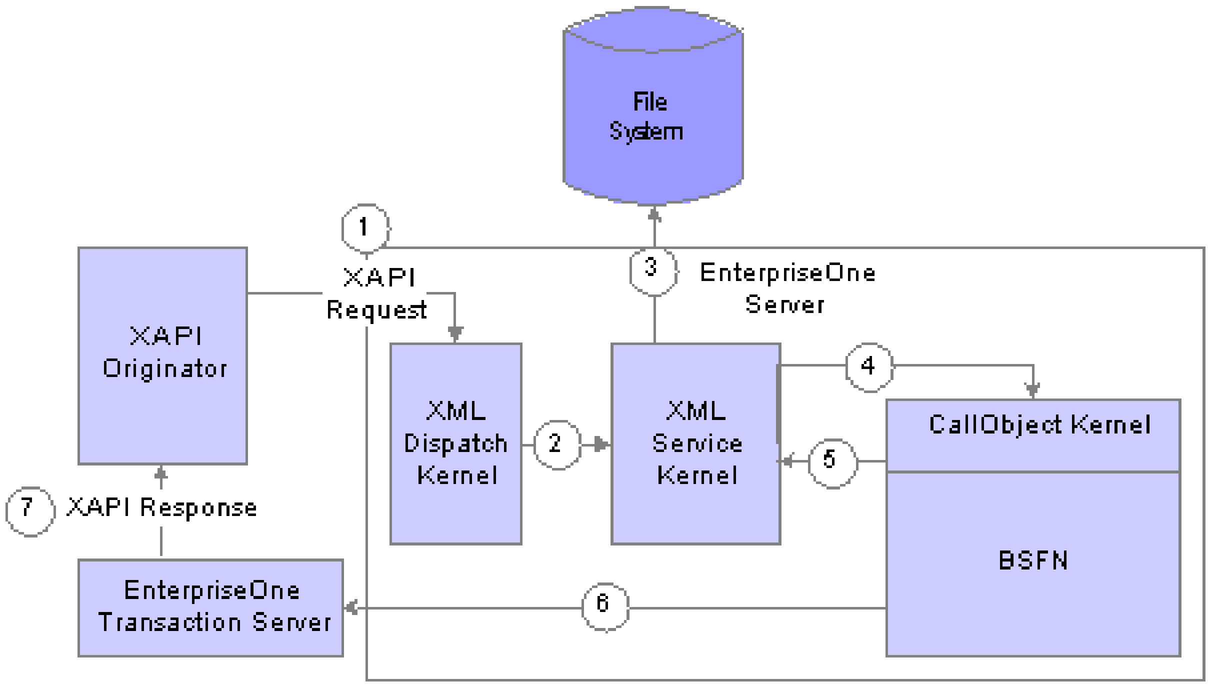
   The CallObject kernel in which the XAPI APIs are executing creates the XAPI request data, adding the callback function. If the XAPI executor is another JD Edwards EnterpriseOne system, the host and port of the JD Edwards EnterpriseOne server that is functioning as the XAPI originator is added to the 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. 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.

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.



In summary:

1. 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. 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.

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.

> **Note:** JD Edwards EnterpriseOne Tools API Reference Guide *https://support.oracle.com/rs?type=doc;id=20705446.1*

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.

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

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
```

XAPI Inbound Response APIs

These APIs are available for you to read an inbound XAPI response:

- jdeXML_GetDSCount
- jdeXML_GetDSName
- jdeXML_ParseDS
- jdeXML_DeleteXML

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
```

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.

> **Note:** JD Edwards EnterpriseOne Tools API Reference Guide ***https://support.oracle.com/rs?type=doc;id=705446.1.***

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).

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.

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).

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.

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

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;
```

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

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);
   }
```

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.

Mapping a Business Function

This section provides an overview of mapping business functions and discusses how to add mapping information.

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.

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.

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.

17 Using Guaranteed Z Events

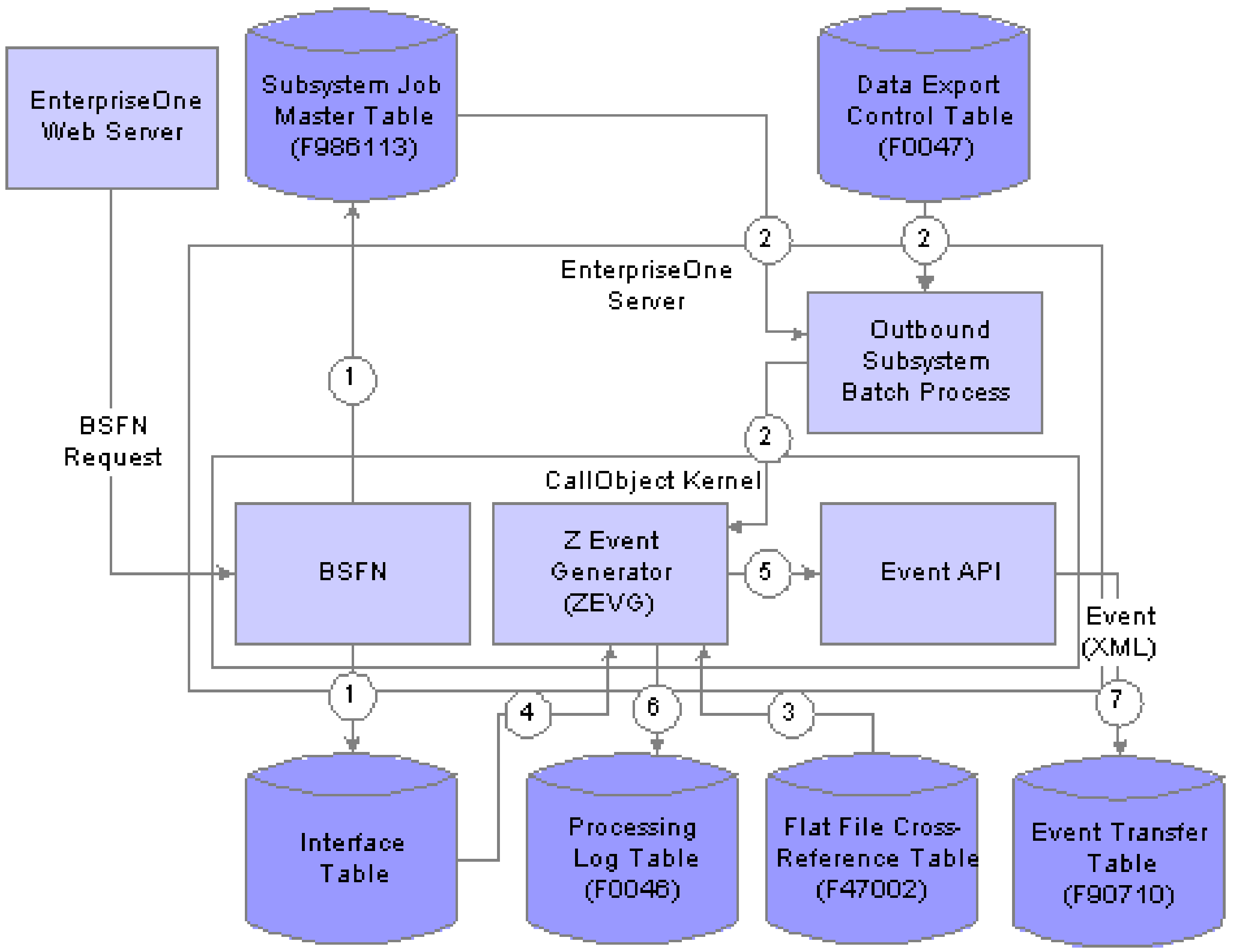
Understanding Guaranteed Z Events

This chapter provides overviews of Z events, the Z event process, and vendor-specific outbound functions, and discusses how to work with Z events.

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 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).


EnterpriseOne Web Server
Subsystem Job Master Table (F986113)
Data Export Control Table (F0047)
EnterpriseOne Server
Outbound Subsystem Batch Process
BSFN Request
CallObject Kernel
BSFN
Z Event Generator (ZEVG)
Event API
Event (XML)
1
2
3
4
5
6
7
Interface Table
Processing Log Table (F0046)
Flat File Cross-Reference Table (F47002)
Event Transfer Table (F90710)

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.

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

Working With Z Events

This section provides an overview about Z event configuration and discusses how to add a data export control record.

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.

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.

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.

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.

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* .

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*.

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 your system, 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.

Setting Up Data Export Controls

This section provides an overview of setting up data export controls and discusses setting up the record.

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.

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.

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 32 bit: _CallOnUpdate@36

     Windows 64 bit : CallOnUpdate

     UNIX: CallOnUpdate

     IBM i : CallOnUpdate
   - Function Library

     Windows: EnterpriseOne Bin32 Path\zevg.dll

     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

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 remove any 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.

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.

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.

Note:

- *Understanding Z Transactions*.

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.

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.

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)

Note:

- *Understanding Guaranteed Z Events*.

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.

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.

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 the *JD Edwards EnterpriseOne Applications Data Interface for Electornic Data Interchange Implementation Guide*

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.

Note:

- *"Understanding Table Conversion" in the JD Edwards EnterpriseOne Tools Table Conversion Guide* .

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.

Note:

- *"Understanding OSA" in the JD Edwards EnterpriseOne Tools Report Printing Administration Technologies Guide* .

19 Using Open Data Access

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
- Microsoft Analysis Service (not certified)

ODA sits between the front-end Query and Reporting tool and the JD Edwards EnterpriseOne-configured ODBC drivers.

Installing ODA

To access JD Edwards EnterpriseOne data with the ODA ODBC driver, your system must meet the minimum technical requirements for JD Edwards EnterpriseOne. Minimum technical requirements are updated for each release. See document 745831.1 (JD Edwards EnterpriseOne Minimum Technical Requirements Reference) on My Oracle Support.

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

Before you install ODA, ensure that your system meets the specified hardware and software requirements.

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).

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.

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.

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.
Data Source	The data that you want to access, as well as the operating system, DBMS, and network platform for the data.

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.

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.
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.

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.

Deleting a Data Source

To delete a data source, access the appropriate data source, select remove, and click Yes to confirm the delete.

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.

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

Option Name	Description
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.
Tables and Business Views	Select this option to view both JD Edwards EnterpriseOne tables and JD Edwards EnterpriseOne business views.

Working with ODA

This section discusses how to:

- Manipulate data.
- Use keywords in the connection string.
- Run a query using Microsoft Excel.

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 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.
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.

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.
SHIFTDECIMALS	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.
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.

Key	Value	Description	Input Connection String	Output Connection String
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.
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. This example shows how to write a connection string:

```
DSN=OneWorld ODA;DBQ=ADEVHP02;
```

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.

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.

Error Message	Description
	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.
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.

Error Message	Description
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.
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.

Error Message	Description
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.
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

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.
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.

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 .

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.

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.

The Type 3 driver can support approximately 1,000 desktops.

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.

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.

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* .

See *"Install a JD Edwards EnterpriseOne Data Access Driver" in the JD Edwards EnterpriseOne Tools Server Manager Guide JD Edwards EnterpriseOne Tools Server Manager Guide*

Note: 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 .

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.

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
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.

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>`

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.

Establishing Connection and Execute Query Using JDBC Driver

This example code shows how to connect and execute query using JDBC Driver:

```
String url = "jdbc:oracle:enterpriseone://JDV920"; String user = “JDE’;
String password = “JDE”;
String query = "SELECT COUNT(DISTINCT SAL),AVG(SAL),HMCO FROM F060116 GROUP BY HMCO";
Connection conn = null; ResultSet rs = null;

try{
JDBCDriver driver = new JDBCDriver();
conn = driver.connect(url, user, password); rs = driver.executeQuery(conn, query);
---
---
---
} catch (SQLException e) {
}finally{
if(rs != null) rs.close();
if (conn != null) conn.close();
}
```

Starting with Tools Release 9.2.7, you can connect to ERP environments with token-based authentication. The example code shows how to connect and execute query using token-based authentication with JDBC Driver.

```
String url = "jdbc:oracle:enterpriseone://JDV920"; String user = "JDE’;
String token = “XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX’;
String query = "SELECT COUNT(DISTINCT SAL),AVG(SAL),HMCO FROM F060116 GROUP BY HMCO";
Connection conn = null; ResultSet rs = null;

try{
JDBCDriver driver = new JDBCDriver();
conn = driver.connectUsingToken(url, user, token); rs = driver.executeQuery(conn, query);
---
---
---
} catch (SQLException e) {
}finally{
if(rs != null) rs.close();
if (conn != null) conn.close();
}
```

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.

   Note: 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.

SQL Grammar

The JD Edwards EnterpriseOne JDBC drivers support different flavors of SQL depending on the connection mode.

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

Rule	Definition
	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*)*
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

Rule	Definition
	| 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

Rule	Definition
elements	*element* (, *element*)*
element	*field-reference* | *literal*
comparison-op	= | != | <> | > | >= | < | <= | *= | =* | *=*

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`

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.)

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.

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.

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.

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.

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 HTML web client.

21 Setting Up Orchestration Cross-References

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.

You also use P952000 to create orchestration cross-reference and white list records for an Internet of Things (IoT) configuration with the EnterpriseOne AIS Server. When creating these types of records, use the AIS category described below.

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.
- AIS

  An AIS reference is used to define key-value data pairs for an IoT orchestration cross-reference or white list. The records that you define in P952000 must match the key-value data pairs defined in the cross-reference and white list XMLs. For more information about IoT orchestration cross-reference and white lists, see "Configuring Cross-Reference XMLs" and "Configuring White List XMLs" in the *JD Edwards EnterpriseOne Tools Orchestrator Guide for Studio Version 8 and Prior* .

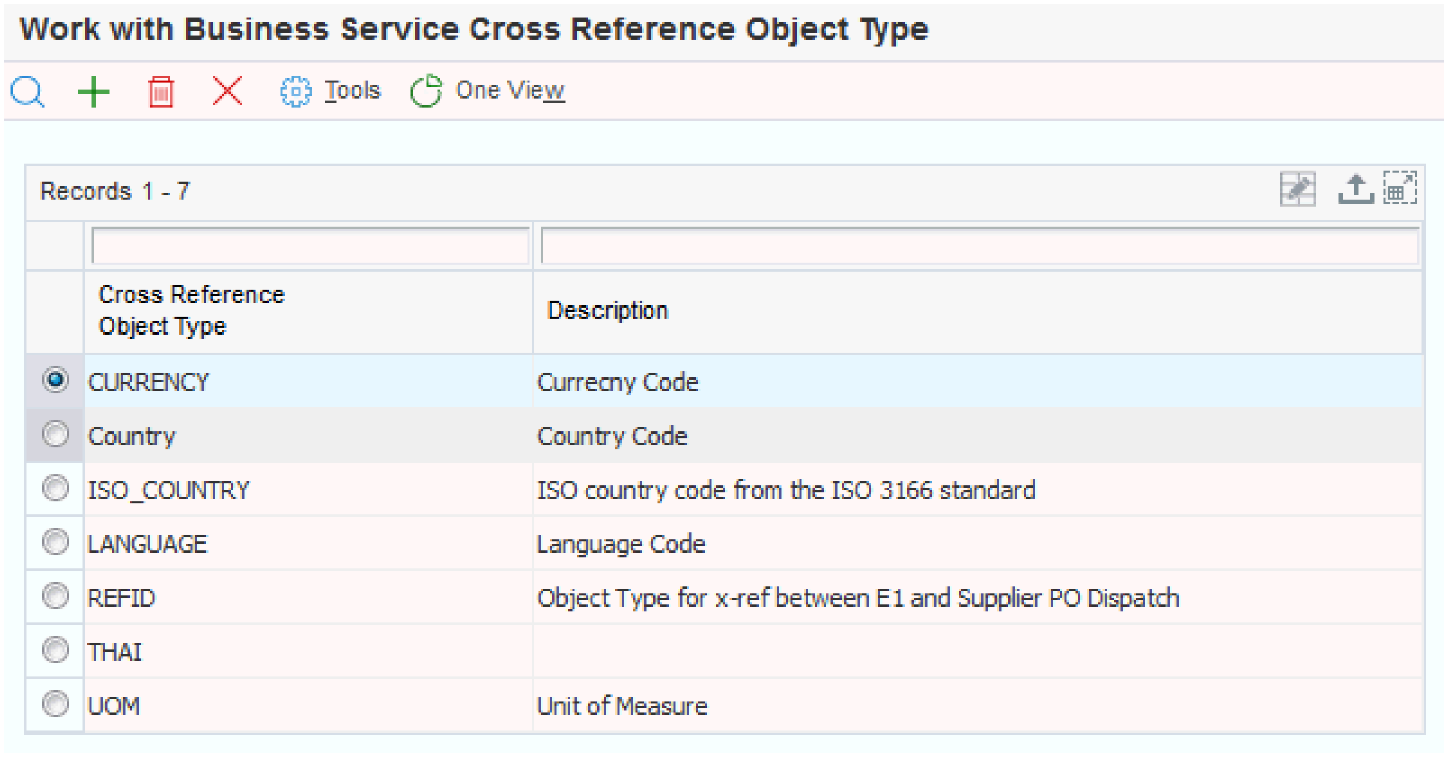
Cross-Reference Categorization

JD Edwards EnterpriseOne uses cross-reference object types to categorize cross-references. You use cross-reference object types to group together code, key, or AIS 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.

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.



	Cross Reference Object Type	Description
●	CURRENCY	Currecny Code
○	Country	Country Code
○	ISO_COUNTRY	ISO country code from the ISO 3166 standard
○	LANGUAGE	Language Code
○	REFID	Object Type for x-ref between E1 and Supplier PO Dispatch
○	THAI	
○	UOM	Unit of Measure

To add a cross-reference object type:

1. To access the Work with Orchestration Cross Reference form, enter **P952000** in the Fast Path field.

   You can access the work with Orchestration Cross Reference form from the EnterpriseOne Navigator using the following path:

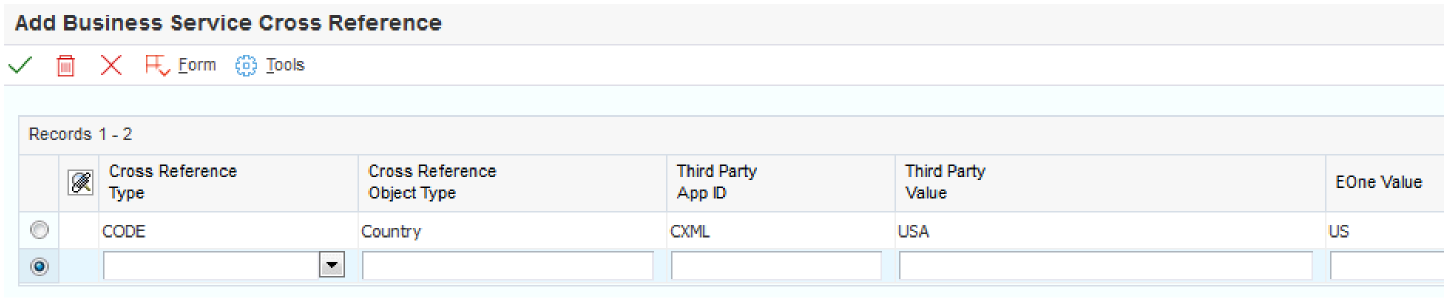
   EnterpriseOne Menus > EnterpriseOne Life Cycle Tools > System Administration Tools > Business Service Property and Business Service Cross Reference Administration > Business Service Cross Reference
2. From the Form menu, select Object Type.
3. On the Work with Orchestration Cross Reference Object Type 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.

Adding Orchestration Cross-References

You add orchestration cross-references to assign JD Edwards EnterpriseOne values to values in a third-party application.

If you are creating IoT orchestration cross-reference or white list records, see *"Setting Up Cross-References and White Lists in EnterpriseOne" in the JD Edwards EnterpriseOne Tools Orchestrator Guide for Studio Version 8 and Prior* for supplemental steps.



Records 1 - 2

Cross Reference Type	Cross Reference Object Type	Third Party App ID	Third Party Value	EOne Value
CODE	Country	CXML	USA	US

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.

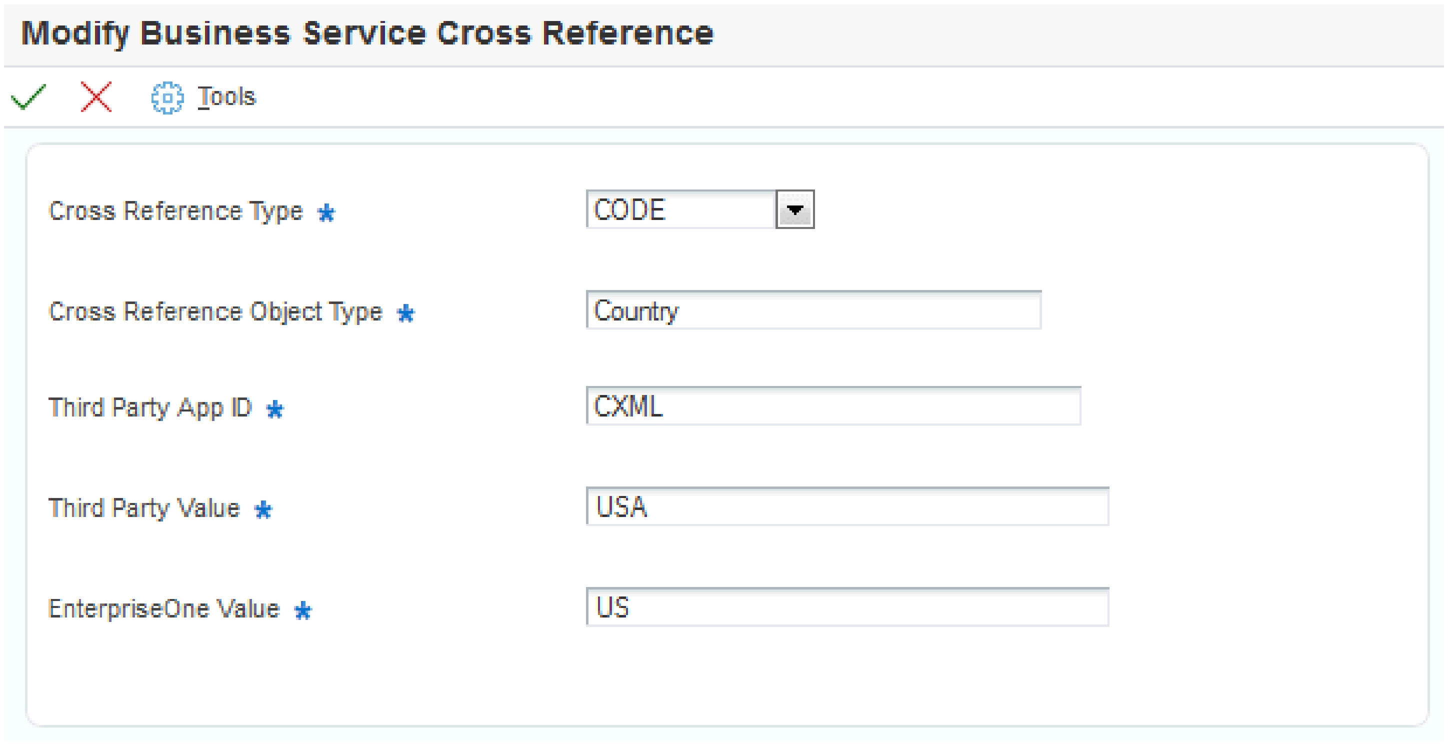
b. Press the Tab key to add additional cross-references as needed, and then click the OK button when complete.

   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.

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.



Cross Reference Type *	CODE
Cross Reference Object Type *	Country
Third Party App ID *	CXML
Third Party Value *	USA
EnterpriseOne Value *	US

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

Deleting Orchestration Cross-References

If a cross-reference becomes obsolete and is no longer necessary, you can delete it.

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

22 Appendix A - Interoperability Interface Table Information

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	R03B11Z1A	R03B11Z1A - 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

Program	Interface Table (Z table)	Input Subsystem Batch Process	Input Processor Batch Process	Extraction Batch Process	Revisions Program	Purge Batch Process	Program with POs
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
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 The table is updated during PO Receipts, Receipts Reversal, and Receipt Routing.
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

Program	**Interface Table (Z table)**	**Input Subsystem Batch Process**	**Input Processor Batch Process**	**Extraction Batch Process**	**Revisions Program**	**Purge Batch Process**	**Program with POs**
Manufacturing	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
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

Program	Interface Table (Z table)	Input Subsystem Batch Process	Input Processor Batch Process	Extraction Batch Process	Revisions Program	Purge Batch Process	Program with POs
							(Each done individually)
Kanban Transactions	F30161Z1	R30161Z1I - XJDE0002	R30161Z1I - XJDE0001	N/A	P30161Z1	R30161Z1P	N/A

23 Appendix B - XML Format Examples (All Parameters)

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>
```

Inbound XML Transaction Request and Response Format

This section provides example request and response code that illustrate the inbound XML format with all of the parameters.

Request

This format shows an XML Transaction update request for an inbound purchase order:

```
<?xml version="1.0" encoding="utf-8" ?>
- <!--  This an Inbound Purchase Order.
  -->
- <jdeRequest pwd="password" role="*ALL" type="trans" user="user"
 environment="environment">
- <transaction type="JDEPOIN" action="inbound">
- <key>
  <column name="EdiUserId">TEST</column>
  <column name="EdiTransactNumber">1995598</column>
  <column name="EdiBatchNumber">11004</column>
  <column name="EdiLineNumber">1.000</column>
  </key>
- <!--  table name attribute value is optional, cross reference is used
  -->
- <table name="F4301Z1" type="header">
  <column name="EdiDocumentType" />
  <column name="TypeTransaction">JDEPOIN</column>
  <column name="EdiTranslationFormat" />
  <column name="EdiTransmissionDate" />
  <column name="DirectionIndicator">1</column>
  <column name="EdiDetailLinesProcess">0</column>
  <column name="EdiSuccessfullyProcess">N</column>
  <column name="TradingPartnerId" />
  <column name="TransactionAction">A</column>
  <column name="CompanyKeyOrderNo">00200</column>
  <column name="DocumentOrderInvoiceE">203</column>
  <column name="OrderType">OP</column>
  <column name="OrderSuffix">000</column>
  <column name="CostCenter">27</column>
  <column name="CompanyKeyOriginal" />
  <column name="OriginalPoSoNumber" />
  <column name="OriginalOrderType" />
  <column name="CompanyKeyRelated" />
  <column name="RelatedPoSoNumber" />
  <column name="RelatedOrderType" />
  <column name="AddressNumber">1620</column>
  <column name="AddressNumberShipTo">27</column>
  <column name="DateRequestedJulian">11/10/99</column>
```

```
<column name="DateTransactionJulian">11/10/99</column>
<column name="DateOriginalPromisde">11/10/99</column>
<column name="ActualShipDate" />
<column name="CancelDate" />
<column name="DatePriceEffectiveDate" />
<column name="DatePromisedShipJu" />
<column name="DatePromisedShipJu" />
<column name="Reference1" />
<column name="Reference2Vendor" />
<column name="DeliveryInstructLine1" />
<column name="DeliveryInstructLine2" />
<column name="NameRemark" />
<column name="Description" />
<column name="PrintMessage1" />
<column name="PriceAdjustmentScheduleN" />
<column name="PricingGroup" />
<column name="PaymentTermsCode01" />
<column name="TaxExplanationCode1" />
<column name="TaxArea1" />
<column name="CertificateTaxExempt" />
<column name="HoldOrdersCode" />
<column name="AssociatedText" />
<column name="InvoiceCopies">0</column>
<column name="NatureOfTransaction" />
<column name="ContainerID" />
<column name="FreightHandlingCode" />
<column name="ZoneNumber" />
<column name="BuyerNumber">0</column>
<column name="CarrierNumber">0</column>
<column name="ModeOfTransport" />
<column name="ConditionsOfTransport" />
<column name="ReasonCode" />
<column name="FreightCalculatedYN" />
<column name="ApplyFreight">Y</column>
<column name="PostQuantities">1</column>
<column name="AmountOrderGross">32.10</column>
<column name="PercentRetainage1">0</column>
<column name="RetainageRule" />
<column name="UnitOfMeasureWhtDisp" />
<column name="UnitOfMeasureVolDisp" />
<column name="PurgeCode1" />
<column name="LogicControl" />
<column name="ProcessingMode" />
<column name="TypeMatch" />
<column name="StatusPurchaseOrder" />
<column name="CodeAutomaticVoucher">N</column>
<column name="PrepaymentYN" />
<column name="CorrespondenceMethod" />
<column name="PurchasingReportCode5" />
<column name="RoutingApproval">DEMO</column>
<column name="NumberChangeOrder">0</column>
<column name="CurrencyMode">D</column>
<column name="CurrencyCodeFrom">USD</column>
<column name="CurrencyConverRateOv">0</column>
<column name="LanguagePreference" />
<column name="AmountForeignOpen">0</column>
<column name="OrderedBy">LH5813131</column>
<column name="OrderTakenBy" />
<column name="UserReservedCode" />
<column name="UserReservedDate" />
```

```
  <column name="UserReservedAmount">0</column>
  <column name="UserReservedNumber">0</column>
  <column name="UserReservedReference" />
  <column name="TransactionOriginator" />
  <column name="UserId">LH5813131</column>
  <column name="ProgramId">EP4310</column>
  <column name="WorkStationId">HUANGL1</column>
  <column name="DateUpdated">11/10/99</column>
  <column name="TimeOfDay">100707</column>
  <column name="ReferenceUCISNo" />
  <column name="RequestedshipTime" />
  <column name="AddressNumberMarkfor" />
  <column name="RequestedDeliveryTime" />
  <column name="DocumentOrderInvoi" />
  <column name="DoctType" />
  <column name="AddressNumberBillTp" />
  <column name="CurrencyCodeBase" />
  </table>
- <table name="F4311Z1" type="detail">
  <column name="EdiDocumentType" />
  <column name="TypeTransaction">JDEPOIN</column>
  <column name="EdiTranslationFormat" />
  <column name="EdiTransmissionDate" />
  <column name="DirectionIndicator">1</column>
  <column name="EdiDetailLinesProcess">0</column>
  <column name="EdiSuccessfullyProcess">N</column>
  <column name="TradingPartnerId" />
  <column name="TransactionAction">A</column>
  <column name="DocumentOrderInvoiceE">203</column>
  <column name="CompanyKeyOrderNo">00200</column>
  <column name="OrderType">OP</column>
  <column name="OrderSuffix">000</column>
- <!--  default line number is 1
  -->
- <!--  have to specify line number if multiple records exist
  -->
  <column name="LineNumber">1.000</column>
  <column name="CostCenter">30</column>
  <column name="Company">00001</column>
  <column name="CompanyKeyOriginal" />
  <column name="OriginalPoSoNumber" />
  <column name="OriginalOrderType" />
  <column name="OriginalLineNumber">0</column>
  <column name="CompanyKeyRelated" />
  <column name="RelatedPoSoNumber" />
  <column name="RelatedOrderType" />
  <column name="RelatedPoSoLineNo">0</column>
  <column name="ContractNumberDistributi" />
  <column name="ContractSupplementDistri">0</column>
  <column name="ContractBalancesUpdatedY" />
  <column name="AddressNumber">1620</column>
  <column name="AddressNumberShipTo">1620</column>
  <column name="DateRequestedJulian">11/10/99</column>
  <column name="DateTransactionJulian">11/10/99</column>
  <column name="DateOriginalPromisde">11/10/99</column>
  <column name="ActualShipDate">11/10/99</column>
  <column name="CancelDate" />
  <column name="DatePriceEffectiveDate" />
  <column name="DatePromisedShipJu" />
  <column name="DatePromisedShipJu" />
```

```
<column name="DateServiceCurrency" />
<column name="DtForGLAndVouch1">11/10/99</column>
<column name="PeriodNoGeneralLedge">0</column>
<column name="Reference1" />
<column name="Reference2Vendor" />
<column name="IdentifierShortItem">60003</column>
<column name="Identifier2ndItem">1001</column>
<column name="Identifier3rdItem">1001</column>
<column name="Location" />
<column name="Lot" />
<column name="FromGrade" />
<column name="ThruGrade" />
<column name="FromPotency">0</column>
<column name="ThruPotency">0</column>
<column name="DescriptionLine1">Bike Rack - Trunk Mount</column>
<column name="DescriptionLine2" />
<column name="LineType">S</column>
<column name="StatusCodeNext">280</column>
<column name="StatusCodeLast">230</column>
<column name="ItemNumberRelatedKit" />
<column name="ReportingCode1Sales" />
<column name="ReportingCode2Sales" />
<column name="ReportingCode3Sales" />
<column name="ReportingCode4Sales">444</column>
<column name="ReportingCode5Sales">158</column>
<column name="ReportCode1Purchasing" />
<column name="ReportCode2Purchasing" />
<column name="ReportCode3Purchasing" />
<column name="ReportCode4Purchasing">240</column>
<column name="ReportCode5Purchasing" />
<column name="UnitOfMeasureAsInput">EA</column>
<column name="UnitsTransactionQty">1</column>
<column name="UnitsChangeOrderQty">0</column>
<column name="UnitsOpenQuantity">1</column>
<column name="UnitsLineItemQtyRe">0</column>
<column name="CumulativeReceived">0</column>
<column name="UnitsRelieved">0</column>
<column name="OtherQuantity12" />
<column name="PurchasingUnitPrice">32.1000</column>
<column name="AmountExtendedPrice">32.10</column>
<column name="AmountChange">0</column>
<column name="AmountOpen1">32.10</column>
<column name="AmountReceived">0</column>
<column name="AmountRelieved">0</column>
<column name="TaxCommitment">0</column>
<column name="TaxAmountRelieved">0</column>
<column name="PriceOverrideCode" />
<column name="UnitCostPurchasing">32.1000</column>
<column name="AmountExtendedCost">32.10</column>
<column name="CostOverrideCode" />
<column name="CostMethodPurchasing" />
<column name="PrintMessage1" />
<column name="PriceAdjustmentScheduleN" />
<column name="PricingCategory" />
<column name="PricingCategoryLevel1" />
<column name="CatalogName" />
<column name="DiscountFactor">1.0000</column>
<column name="PaymentTermsCode01" />
<column name="TaxableYN1">N</column>
<column name="TaxExplanationCode1" />
```

```
<column name="TaxArea1" />
<column name="AssociatedText" />
<column name="ContainerID" />
<column name="CommodityCode" />
<column name="NatureOfTransaction" />
<column name="FreightHandlingCode" />
<column name="FreightCalculatedYN">N</column>
<column name="ZoneNumber" />
<column name="RateCodeFrieghtMisc" />
<column name="RateTypeFreightMisc" />
<column name="BuyerNumber">0</column>
<column name="CarrierNumber">0</column>
<column name="ModeOfTransport" />
<column name="ConditionsOfTransport" />
<column name="ShippingCommodityClass" />
<column name="ShippingConditionsCode" />
<column name="UnitOfMeasurePrimary">EA</column>
<column name="UnitsPrimaryQtyOrder">1</column>
<column name="UnitOfMeasureSecondary">EA</column>
<column name="UnitsSecondaryQtyOr">1</column>
<column name="UnitOfMeasurePurchas">EA</column>
<column name="AmountUnitWeight">80.0000</column>
<column name="WeightUnitOfMeasure">OZ</column>
<column name="AmountUnitVolume">2.2500</column>
<column name="VolumeUnitOfMeasure">FC</column>
<column name="GlClass">IN30</column>
<column name="Century">0</column>
<column name="FiscalYear1">0</column>
<column name="LineStatus" />
<column name="ReasonCode" />
<column name="ApplyFreight" />
<column name="PostQuantities" />
<column name="GrossWeight">0</column>
<column name="UnitOfMeasureGrossWt" />
<column name="LedgerType" />
<column name="AcctNoInputMode" />
<column name="AccountId" />
<column name="PurchasingCostCenter" />
<column name="ObjectAccount" />
<column name="Subsidiary" />
<column name="SubledgerType" />
<column name="Subledger" />
<column name="SerialTagNumber" />
<column name="CostComponentNumber">0</column>
<column name="TagReference" />
<column name="CategoriesWorkOrder001" />
<column name="Plan" />
<column name="Elevation" />
<column name="CategoryCodeGl001" />
<column name="RetainageRule" />
<column name="CodeLocationTaxStat" />
<column name="PurgeCode1" />
<column name="ProcessingMode" />
<column name="FinalPayment" />
<column name="CodeAutomaticVoucher">N</column>
<column name="PrepaymentYN" />
<column name="WoOrderFreezeCode">N</column>
<column name="TypeMatch" />
<column name="RoutingProcessYN" />
<column name="ReceiptCode" />
```

```
  <column name="PurchaseOrderStatus01" />
  <column name="PurchaseOrderStatus02" />
  <column name="PurchaseOrderStatus03" />
  <column name="PurchaseOrderStatus04" />
  <column name="PurchaseOrderStatus05" />
  <column name="PurchaseOrderStatus06" />
  <column name="PurchaseOrderStatus07" />
  <column name="PurchaseOrderStatus08" />
  <column name="PurchaseOrderStatus09" />
  <column name="PurchaseOrderStatus10" />
  <column name="CorrespondenceMethod" />
  <column name="RoutingApproval">DEMO</column>
  <column name="NumberChangeOrder">0</column>
  <column name="ChangeOrderType" />
  <column name="DocumentChangeOrder">0</column>
  <column name="ChangeOrderLineNumber">0</column>
  <column name="CurrencyCodeFrom">USD</column>
  <column name="CurrencyConverRateOv">0</column>
  <column name="ForeignPurchasingCost">0</column>
  <column name="AmountForeignExtPrice">0</column>
  <column name="AmountForeignUnitCost">0</column>
  <column name="AmountForeignExtCost">0</column>
  <column name="ForeignChangedAmount">0</column>
  <column name="AmountForeignOpen">0</column>
  <column name="AmountReceivedForeign">0</column>
  <column name="UserReservedCode" />
  <column name="UserReservedDate" />
  <column name="UserReservedAmount">0</column>
  <column name="UserReservedNumber">0</column>
  <column name="UserReservedReference" />
  <column name="TransactionOriginator">LH5813131</column>
  <column name="UserId">LH5813131</column>
  <column name="ProgramId" />
  <column name="WorkStationId">HUANGL1</column>
  <column name="DateUpdated">11/10/99</column>
  <column name="TimeOfDay">100712</column>
  </table>
- <inboundUBE>
  <UBEName>R4311Z1I</UBEName>
  <UBEVersion>XJDE0002</UBEVersion>
  </inboundUBE>
- <callbackFunction>
  <functionName />
  <functionLibrary />
  </callbackFunction>
  </transaction>
  </jdeRequest>
```

Response

This format shows the XML Transaction update response for an inbound purchase order:

```
 <?xml version="1.0" encoding="UTF-8" ?>
- <jdeResponse role="*ALL" type="trans" user="user" xmlns="urn:Schemas-jdedwards-
com:trans.response.JDEPOIN" environment="environment">
- <transaction type="JDEPOIN" action="inbound">
  <returnCode code="0">XML Request OK</returnCode>
```

```
- <key>
  <column name="EdiUserId">TEST</column>
  <column name="EdiTransactNumber">2995598</column>
  <column name="EdiBatchNumber">11004</column>
  <column name="EdiLineNumber">1.000</column>
  </key>
  <writeSubsystemRecord>SUCCESS</writeSubsystemRecord>
  </transaction>
  </jdeResponse>
```

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.

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>
```

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>
```

24 Appendix C - Minimum Required Values Sample Code

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>
```

25 Appendix D - XML Format Examples (Events)

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>
```

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

26 Glossary

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.

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.

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.

connection mode

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.

correlation data

The data used to tie HTTP responses with requests that consist of business service name and method.

cross-reference utility services

Utility services installed in a BPEL/ESB environment that access EnterpriseOne cross-reference data.

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.

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 extension

A JDeveloper component (plug-in) specific to EnterpriseOne. A JDeveloper wizard

is a specific example of an extension.

JMS Queue

A Java Messaging service queue used for point-to-point messaging.

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.

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.

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.

real-time event

A message triggered from EnterpriseOne application logic that is intended for external systems to consume.

SOA

Abbreviation for Service Oriented Architecture.

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.

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 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