JD Edwards EnterpriseOne Tools

Business Services Development Guide

9.2

JD Edwards EnterpriseOne Tools
Business Services Development Guide

9.2

Part Number: E53611-04

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 Business Services Development

JD Edwards EnterpriseOne Tools Business Services Development Overview

Oracle's JD Edwards EnterpriseOne Tools Business Services Development provides guidelines for creating JD Edwards EnterpriseOne web services. This document provides information for creating web services for JD Edwards EnterpriseOne to be both a provider and consumer of web services. This document identifies the tools for creating business services for JD Edwards EnterpriseOne web service interoperability.

> **Note:** Oracle reserves the right to reorganize the business services foundation packages (jar files) for tools release upgrades. If you are planning to upgrade your system, test your custom objects and modify them as appropriate to ensure your code will continue to work as intended. You cannot upgrade custom business service objects after you install a tools release upgrade.

JD Edwards EnterpriseOne Tools Business Services Development Implementation

This section provides an overview of the steps that are required to implement JD Edwards EnterpriseOne Tools Business Services Development tools.

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 preformed before developing JD Edwards EnterpriseOne business services:

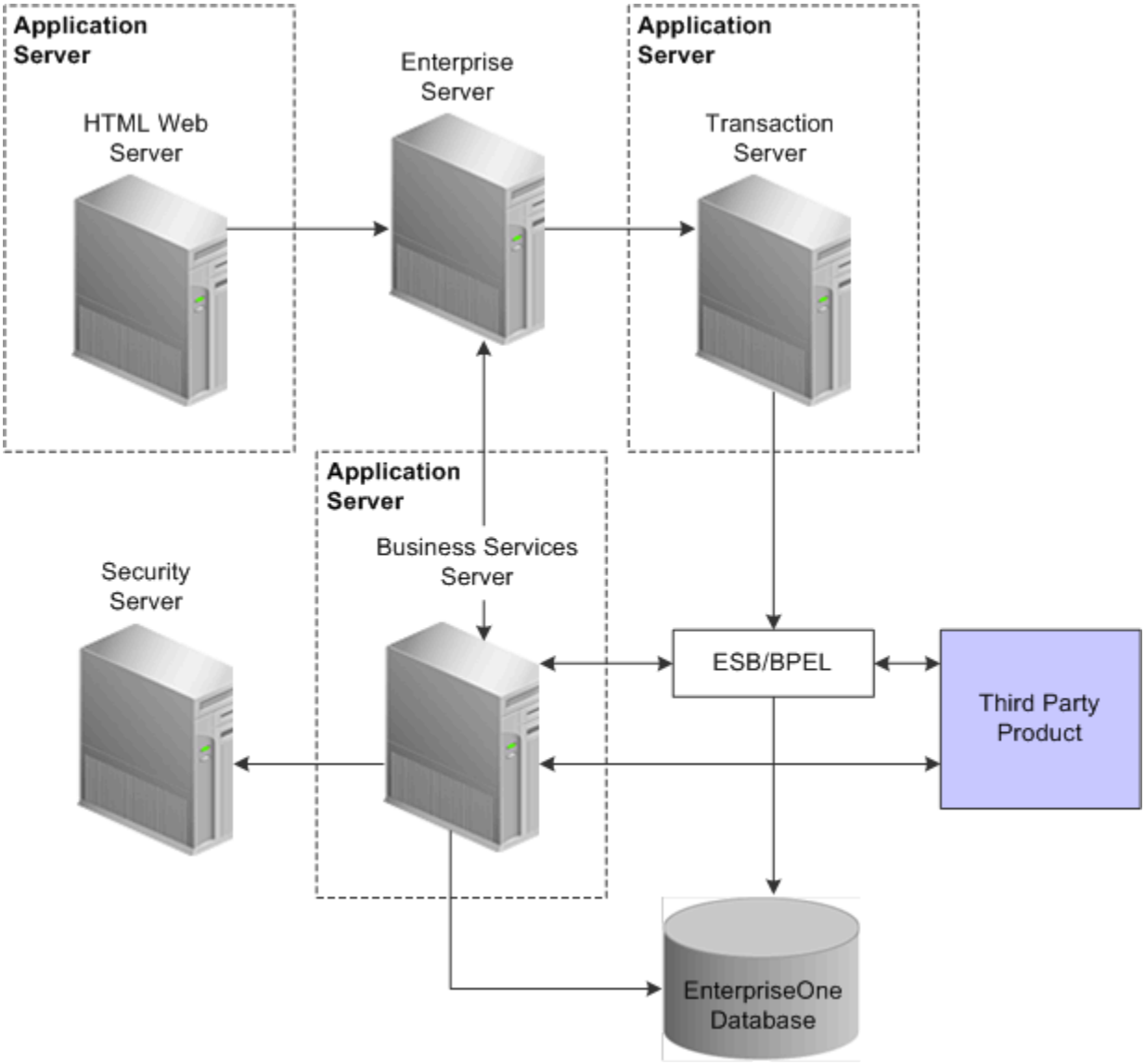
1. Install EnterpriseOne Tools.
   See *JD Edwards EnterpriseOne Tools Server Manager Guide*
2. Configure the Business Services Server.
   See *"Configuring the Business Services Server" in the JD Edwards EnterpriseOne Tools Business Services Server Reference Guide*
3. Deploy the business services packages to the Business Services Server.
   See *"Understanding Packages for Business Services" in the JD Edwards EnterpriseOne Tools Package Management Guide*
4. Create business services workspace and projects on JDeveloper using Object Management Workbench.
   See *"Working with Business Services" in the JD Edwards EnterpriseOne Tools Object Management Workbench Guide*

2 Understanding Business Services Development

JD Edwards EnterpriseOne Business Services Architecture

JD Edwards EnterpriseOne provides interoperability with other Oracle products and third-party products and systems by natively producing and consuming web services. Web services are standardized ways to interoperate among disparate enterprises. JD Edwards EnterpriseOne web services conform to industry standards. JD Edwards EnterpriseOne web services use a Service-Oriented Architecture (SOA) environment to provide and consume inbound transactions and to send outbound event notifications.

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



JD Edwards EnterpriseOne as a Web Service Provider

As a web service provider, JD Edwards EnterpriseOne exposes web services for consumption by external systems. Each operation of the web service performs a business process. Multiple Java classes are used to perform the requested business process. The JD Edwards EnterpriseOne 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 the JD Edwards EnterpriseOne system. 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.

JD Edwards EnterpriseOne as a 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 an external web service. A web service proxy is provided end point and security information for the external web service. The external web service returns the results of the call to the business service. The business service passes the results to the business function.

You also can call external services using XML over HTTP when a web service is not available.

Business Services Server

The business services product provides a development environment and a production environment. Integration developers create and test new business services using the development environment. Orchestration developers use the production environment to develop orchestrations including published business service classes, which are exposed as web services.

Development Environment

The business services server provides a business services development client for developing and testing business services as both a web service provider and a web service consumer. The business services development client provides Oracle's JDeveloper and JD Edwards EnterpriseOne Business Services Framework as tools to help you create business services. JDeveloper has wizards and code templates that provide Java code to help you develop business services that conform to web service standards. Tools are provided on the development client to enable you to test all aspects of business service development in an isolated environment prior to checking in your code. You must install JDeveloper on your development client.

Each business service class has compile and runtime dependencies on the business services framework classes. The business services framework consists of base classes from which other classes extend. The business service framework provides services such as business function processing and database access.

Reference implementations are included in your software delivery. The business service reference implementations are sample business services provided by JD Edwards EnterpriseOne. You can use the reference implementations as models for creating custom business services that can be exposed as web services. A reference implementation also shows how to enable JD Edwards EnterpriseOne consuming a web service.

Production Environment

The business services server production environment is used by administrators to create web services. An administrator generates web service interfaces that expose the business service logic. During the web service generation, Web Services Description Language (WSDL) files are produced. WSDL files describe the available web services and are used by external systems during orchestration development. After web services are generated from the Java classes, they can be consumed by external systems.

The production environment also enables JD Edwards EnterpriseOne to consume web services.

3 Understanding the Business Services Server

Business Services Server Overview

The business services server enables JD Edwards EnterpriseOne to natively produce and consume web services. The business services server, which is built on top of a Java 2 Platform, Enterprise Edition (J2EE) server, can be an Oracle WebLogic Server (WLS) or a WebSphere Application Server (WAS). The business services server is required for creating JD Edwards EnterpriseOne provider and consumer web services. Applications that are developed or run on the business services server are written in the Java programming language.

When you install the JD Edwards EnterpriseOne software, the Server Manager deploys an application to the application server. This application contains the business service foundation and business service reference implementations. The business service foundation is a collection of executable files that are required for running business services. The business service reference implementations are sample business services that are provided by JD Edwards EnterpriseOne. You can use the reference implementations as models for creating custom business services that can be exposed as web services. A reference implementation for creating web services for JD Edwards EnterpriseOne as a web service consumer is provided.

Business Service Security

Security for the JD Edwards EnterpriseOne business services consists of two main categories—authorization and authentication.

- Access to run published business services is managed through the JD Edwards EnterpriseOne Security Workbench. For published business services, JD Edwards EnterpriseOne uses a *secure by default* security model, which means that users cannot run a published business service unless a security record exists that authorizes access.
- Authentication for consuming JD Edwards EnterpriseOne business services uses standard JD Edwards EnterpriseOne user credentials or anonymous login.

Business service security ensures that business service consumers are authenticated in JD Edwards EnterpriseOne and are authorized to use the business services. The business services server uses the JD Edwards EnterpriseOne Login Module to ensure that consumers are authenticated. The module uses Java authentication and authorization service (JAAS) to validate the JD Edwards EnterpriseOne users against the EnterpriseOne Security Server. Alternatively, to give non-JD Edwards EnterpriseOne users access to business services, you set up an anonymous user login in the jdbj.ini file.

Note:

- *"Understanding Business Services Server Security" in the JD Edwards EnterpriseOne Tools Business Services Server Reference Guide* .
- *"Managing Published Business Services Security" in the JD Edwards EnterpriseOne Tools Security Administration Guide* .

Business Services Server Scalability

Scalability for the JD Edwards EnterpriseOne business services server depends on scalability features provided by the host application server—Oracle WebLogic Server or WebSphere Application Server.

Business Services Server Fault Tolerance

When a component or machine in the system goes down or is brought down, other components and machines in the system should gracefully degrade and reconnect when the component or machine is back up. A system is considered to be fault-tolerant when these conditions exist:

- Error messages to the user and administrator are meaningful when a component of the system cannot be contacted.
- Connections can be reestablished when a component of the system is restarted without administrative interaction on other components of the system.

The components that are relevant for the business services server to be fault-tolerant are the enterprise server and the security server.

The connection to the enterprise server is fault-tolerant. If the enterprise server is down, the exceptions that are returned from a called web service are descriptive and indicate the problem. When the enterprise server comes back up, subsequent web service calls connect correctly without restarting or any further administration of the business services server. If connections to the enterprise server times out, the connections are reestablished.

The connection from the business services server to the security server is based on a token. If the security server is down or cannot be contacted, the exception message that is returned to the web service caller indicates that the server login failed. When the security server comes back up, the token is validated without administrator interaction.

Business Services Server Clustering Support

You can configure the business services server to support clustering features, such as high availability, scalability, and load balancing, for consumer business services. Business services server clustering enables multiple hosts/ports to receive JDENet messages from the Enterprise Server.

See *"Configuring a Clustered Business Services Server Instance for Consumer Business Services" in the JD Edwards EnterpriseOne Tools Server Manager Guide* .

4 Working with JDeveloper

Understanding JDeveloper

Note: If you are using JD Edwards EnterpriseOne Tools Release 9.2 with JD Edwards EnterpriseOne Applications Release 9.2, JDeveloper 12c is installed on your system. With JDeveloper 12c, you can create only JAX-WS business services. If you are using JD Edwards EnterpriseOne Tools Release 9.2 with JD Edwards EnterpriseOne Applications Release 9.0 or 9.1, JDeveloper 11g is installed on your system. With JDeveloper 11g, you can create JAX-WS and JAX-RPC business services.

Oracle's JDeveloper provides an integrated development environment for creating JD Edwards EnterpriseOne business services. The JD Edwards EnterpriseOne package includes a JDeveloper extension. The extension package contains JD Edwards EnterpriseOne code features that help you create business services. Among these code features are wizards that generate a structure for creating a Java class. Wizards for generating code to call a business function or a data base operation are included in the extension package. The extension package also provides code templates specifically designed to help you develop and test business services. The wizards and code templates help you by enforcing coding conventions so that your business service classes can be exposed as web services.

For JDeveloper 12c, the extension package and plugins are loaded into JDeveloper when you use a feature provided by the extension. For loading EnterpriseOne development extensions, select the option Initialize E1 Workspace under the Tools menu after launching JDeveloper from JD Edwards EnterpriseOne Object Management Workbench (OMW).

Note: If you use JDeveloper 11g, plugins and extensions are automatically installed when you launch JDeveloper from JD Edwards EnterpriseOne Object Management Workbench (OMW).

For JDeveloper to successfully be installed to your development client, you should verify that your system meets the Minimum Technical Requirements (MTRs) for the hardware and software that is required to install JDeveloper. 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

If you use non-English characters or data in your business services, you can configure JDeveloper to support UTF-8.

Note:

- *Understanding UTF-8*.

For more information about Oracle JDeveloper, see the *Oracle Fusion Middleware User's Guide for Oracle JDeveloper.* This document is located under the Fusion Middleware section on the documentation page of the Oracle Technology Network website.

Creating Java Standards-Based Web Services in JDeveloper

JDeveloper 12c supports the Java API for XML-Based Web Services (JAX-WS) standard for creating both provider and consumer business services. JDeveloper 12c supports creating JAX-WS business services only.

Note: If your system has JDeveloper 11g installed, you can create business services for these two web service standards:

- Java API for XML-Based Remote Procedure Calls (JAX-RPC)
- Java API for XML-Based Web Services (JAX-WS)

JDeveloper includes the integrated WebLogic server. After you create a web service, you deploy it to the integrated WebLogic server so that you can test it.

Note:

- *Creating Java Standards-Based Web Services in JDeveloper*
- *Creating a JAX-WS-Based EnterpriseOne Consumer Business Service Using JDeveloper 12c or JDeveloper 11g*

Understanding JAX-RPC Web Services

JAX-RPC is an API for building web services and clients that use remote procedure calls (RPC) and XML. In JAX-RPC, a remote procedure call is represented by an XML-based protocol such as SOAP. The SOAP specification defines the envelope structure, encoding rules, and a convention for representing remote procedure calls and responses. These calls and responses are transmitted as SOAP messages over HTTP. The Web Service Definition Language (WSDL) specifies the XML format for describing the service as a set of endpoints operating on messages.

Note: JD Edwards EnterpriseOne business services use JAX-RPC with SOAP 1.1 over HTTP 1.1 and WSDL 1.1.

Understanding JAX-WS Web Services

JAX-WS is an API for building web services and clients that are message oriented. JAX-WS supports transmission of SOAP messages over HTTP. It also supports sending XML messages over HTTP without SOAP. JAX-WS delegates data binding related tasks to JAXB. JAXB provides support for Java to XML mapping, additional support for less used XML schema constructs, and bidirectional customization of Java and XML data binding.

Note: JD Edwards EnterpriseOne business services use JAX-WS with SOAP 1.1 over HTTP 1.1 and WSDL 1.1. The SOAP message format is Document/Literal.

Viewing JD Edwards EnterpriseOne Code Templates in JDeveloper

The JD Edwards EnterpriseOne code templates are preconfigured in JDeveloper11g. All of the templates you need to generate code and to support business services are provided, though you can add and create new templates if you want to.

To view the code templates, select Tools, Preferences, Code Editor, and then Code Templates.

5 Understanding the Business Services Framework

Business Services Framework

JD Edwards EnterpriseOne provides interoperability with other Oracle applications and third-party systems by natively producing and consuming web services. A web service is a standardized way for disparate systems and applications to exchange information. JD Edwards EnterpriseOne web services are called published business services. A published business service is an Object Management Workbench (OMW) object consisting of one or more classes, one of which publishes methods. Each method performs a business process. You create a web service by creating a published business service and identifying the published class.

JD Edwards EnterpriseOne business services are classes that enable JD Edwards EnterpriseOne to expose a business transaction as a basic web service. A JD Edwards EnterpriseOne business service is an OMW object consisting of one or more classes that expose public methods. Each method performs a business process. These public methods can be called from other business service classes or from published business service classes. Business services are created for internal use. A business service calls a business function or database operation to perform a specific task. Business services are called by a published business service. JD Edwards EnterpriseOne provides reference implementations that you can use as a model for creating your published business services and business services. The reference implementations are for reference only; they are not intended to be used in a production environment.

If you require a feature that is not in a JD Edwards EnterpriseOne published business service, you can extend the existing published business service class by adding functionality before or after the call to the service, or you can create a new published business service. You extend a published business service by creating a new business service and calling the existing published business service from the new service. Oracle recommends that you create a new OMW object (published business service or business service) instead of modifying an existing JD Edwards EnterpriseOne object. This recommendation takes into consideration JD Edwards EnterpriseOne updates and upgrades. During an update or upgrade, any modifications that you made to a JD Edwards EnterpriseOne object will be replaced by the updated or upgraded JD Edwards EnterpriseOne code.

The JD Edwards EnterpriseOne business services framework provides a set of foundation packages. Each foundation package contains a set of interfaces and related classes that provide building blocks that you use to create the published business service or business service.

The JD Edwards EnterpriseOne business services framework includes these packages:

- oracle.e1.bssvfoundation.base
- oracle.e1.bssvfoundation.connection
- oracle.e1.bssvfoundation.exception
- oracle.e1.bssvfoundation.http
- oracle.e1.bssvfoundation.services
- oracle.e1.bssvffoundation.util

The base package contains classes upon which all published business services and business services depend. Key classes and their purposes are described briefly here. The code is commented so that you can generate a Javadoc that provides details of the packages. Appendix C discusses Javadoc.

Each published business service class extends from the PublishedBusinessService class. Processing for a published business service takes place between startPublishedMethod and finishPublishedMethod. When a published business service class is started, the Context class is created. The Context class provides access to properties related to a specific service request as well as resources and behaviors shared between requests. Context is returned from the startPublishedMethod and passed to any called business service. When a published business service class is finished, the finishPublishedMethod commits transactions, handles system logging, and releases the Context class.

To be compliant with Document/Literal SOAP message format, all published business service methods are designed to have only one parameter. This parameter is referred to as the value object. Value object is a class that holds values but has little or no business logic. Published business services receive a value object as a parameter and return another value object with the results of a successful call. A value object used in a published method is referred to as exposed. All value objects extend from the ValueObject class of the base package.

If an error occurs during processing, the published business service method returns an exception message to the caller. This exception message describes the source of the problem. If less severe problems occur (such as ones that issue warnings or informational messages), the call returns successfully, and problems are reported back to the caller in the return value object.

All internal business service classes extend from the BusinessService class. Business service methods should be static and should not contain state information. Business service methods begin processing with a call to the startInternalMethod and complete processing with a call to finishInternalMethod. Unlike published business service methods, business service methods have more than one parameter. Business services methods are passed a Context object, a Connection object, and a ValueObject on which to operate. Also unlike a published business service method, business service methods use the passed value object both for input and to return results. In addition to the values contained within the value object, a business service method conditionally returns an array of E1Message objects, which contains application error messages, warnings, and informational messages to the caller.

The generated Javadoc provides a list and description of the interfaces and classes that are grouped within each of the business service foundation packages. Other pertinent information, such as method and constructor information, is also documented. Appendix C provides more information about Javadoc.

In addition to the business services framework, sample business services that you can use as a reference for creating a new business service are included with the JD Edwards EnterpriseOne software package. You view the sample business services by adding them to an OMW project and opening JDeveloper.

Note: *olink:EOTRI*

- *Understanding Business Services*.
- *Business Services Framework*.
- *"Development Methodology" in the JD Edwards EnterpriseOne Tools Business Services Development Methodology Guide* .
- *JD Edwards EnterpriseOne Tools Interoperability Reference Implementations Guide* .

6 Working with JD Edwards EnterpriseOne as a Web Service Provider

Working with JD Edwards EnterpriseOne as a Web Service Provider

Note: Oracle reserves the right to reorganize the business services foundation packages (jar files) for tools release upgrades. If you are planning to upgrade your system, test your custom objects and modify them as appropriate to ensure your code will continue to work as intended. You cannot upgrade custom business service objects after you install a tools release upgrade.

Understanding JD Edwards EnterpriseOne as a Web Service Provider

Note: If you are using JD Edwards EnterpriseOne Tools Release 9.2 with JD Edwards EnterpriseOne Applications Release 9.2, JDeveloper 12c is installed on your system. With JDeveloper 12c, you can create JAX-WS business services only. If you are using JD Edwards EnterpriseOne Tools Release 9.2 with JD Edwards EnterpriseOne Applications Release 9.0 or 9.1, JDeveloper 11g is installed on your system. With JDeveloper 11g, you can create JAX-WS and JAX-RPC business services.

You use JDeveloper and the Business Services Framework to create published business services and business services. If you are using JDeveloper 12c, do not use APIs that are specific to JDK1.7.

If are using JDeveloper 11g, you can create web services from the published business service using JAX-RPC or JAX-WS web services standards.

Published Business Services

A published business service is an Object Management Workbench (OMW) object consisting of one or more classes. One of the classes is the published business service class, which is the class that publishes methods that are exposed to the public. This public method wraps an internal business service method, where the actual business logic is performed. The published business service contains the value object classes that are received and returned by the published methods. A web service is generated from the published business service class, and the public methods of this class are operations within that web service.

After a business service is published, you cannot change the name and signature of the business service without affecting the consumers of that service. If you change an underlying business service that the published method exposes, then you change the signature and contract of the published business service.

Business Services

A business service is an OMW object consisting of one or more classes. One of the classes is a business service class, which is a Java class that has public methods that are used by other business services and published business services. The methods access logic in JD Edwards EnterpriseOne and support a specific step in a business process. When you create the business service class, you should consider including methods that have similar functionality and manageability in the same business service class. If multiple processes are similar and can reuse code, then these methods should exist in the same business service.

You can modify a business service providing that the change does not alter the signature or behavior of the published business service. You can change a business service in many ways, and how you change the business service depends on the business service design and the type of change that is required. Any change to a business service should be determined as part of the design process. Business service methods can call business functions, database operations, or another business service .

Calling a Business Function

You can create business service methods that call 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 performs a specific task. You use the business service foundation Business Function Call Wizard to create a business function call.

Calling a Database Operation

You can create business service methods that call database operations. Database operations include query, insert, update, and delete. You use the business service foundation Database Call Wizard to create these business services.

Transaction Processing

Transaction processing is a way to update the JD Edwards EnterpriseOne database. A transaction is a logical unit of work performed on the database to complete a common task and maintain data consistency. The database is updated when a transaction is either automatically or manually committed. The business service framework provides two types of default transactions: manual commit connection and auto commit connection.

For a single manual commit transaction, the default behavior is to scope all processing within the published business service method. If any operation within this scope fails, all operations are rolled back, and the published business service method throws an exception. This behavior is recommended when you commit multiple records to multiple tables.

For a single auto commit transaction, the default behavior is for each operation to commit or roll back immediately, which means that each table update within each business function call is either committed or rolled back immediately. This behavior is recommended for queries for which no transaction is needed or when you commit a single record to a single table.

When you are deciding which type of connection to use, you should always consider the business function behavior.

Default transaction behavior should cover most scenarios, but you can define a business service method that explicitly manages transactions. When determining whether a business service requires explicit transaction processing, you should review current JD Edwards EnterpriseOne functionality in the application. If the application uses explicit transaction processing, you should carefully review whether the business service should handle transaction processing the same way.

The *JD Edwards EnterpriseOne Tools Business Services Development Methodology Guide* provides a detailed discussion about transaction processing. In addition, the chapters pertaining to creating a published business service and creating a business service provide an overview of creating a transaction.

See also:

"Auto Commit" in the JD Edwards EnterpriseOne Tools Business Services Development Methodology Guide .

Business Service Properties

Business service properties provide a way for you to change a value in a business service method without changing the method code. A business service property consists of a key and a value. The key is the name of the business service property and cannot be changed. Business service properties are OMW objects. You can use OMW or the Business Service Property Admin program (P951000) to create and maintain them.

Note:

- *Business Service Property Utility Classes*.

Creating a Custom Published Business Service

JD Edwards EnterpriseOne provides reference implementations that you can use as a model for creating a published business service. The reference implementations are for reference only and are not intended to be used in a production environment. The following steps provide how-to information for creating a custom published business service that can be exposed as a web service:

1. Determine whether to create a new published business service or extend an existing published business service.
2. Create a new OMW object for the published business service.

3. Add classes to the published business service object.
    - Create published business service class.
        – Name the published business service class.
        – Create a transaction.
        – Name the published business service methods.
    - Create value object classes.
        – Create business function value object classes.
        – Name the input value object classes.
        – Name the response value object classes.
        – Use valid data types.
    - Create database operation value object classes.
        – Name the input value object classes.
        – Name the response value object classes.
        – Use valid data types.
4. Add business logic.
    - Call business services.
    - Handle errors.
    - Format data.
5. Test the published business service.

Note:

- *Understanding Published Business Service Classes*.
- *"Understanding Published Business Services" in the JD Edwards EnterpriseOne Tools Business Services Development Methodology Guide* .
- *"Understanding Business Services" in the JD Edwards EnterpriseOne Tools Object Management Workbench Guide* .

Testing a Published Business Service

You use tooling provided by JDeveloper to test and debug published business service methods. The published business service class is generated to a web service described in Web Services Description Language (WSDL) format and runs on JDeveloper.

After the published business service is tested as a web service, you verify that the WSDL is compliant. You use JDeveloper for this task.

For more information about Oracle JDeveloper, see the *Oracle Fusion Middleware User's Guide for Oracle JDeveloper.* This document is located under the Fusion Middleware section on the documentation page of the Oracle Technology Network website.

Creating a Custom Business Service

JD Edwards EnterpriseOne provides reference implementations that you can use as a model for creating business services that are specific to your interoperability requirements. The following steps provide how-to information for creating a custom business service:

1. Determine whether to create a new business service or modify an existing business service.
2. Create a new OMW object for the business service.
3. Add classes to the business service object.
    - Create a business service class.
        - Name the business service class.
        - Create a transaction, if necessary (IConnection objects).
        - Declare the business service public methods.
    - Create the internal value object class.
        - Name the internal value object class.
        - Transform data types.
4. Add business service logic.
    - Create a business function call.
    - Create a database call.
    - Call existing business service methods.
    - Use business service properties.
    - Handle errors.
    - Format data.

Note:

- *Understanding Business Service Classes*.
- *"Understanding Business Services" in the JD Edwards EnterpriseOne Tools Business Services Development Methodology Guide* .
- *"Understanding Business Services" in the JD Edwards EnterpriseOne Tools Object Management Workbench Guide* .

Creating Java Standards-Based Web Services in JDeveloper

JDeveloper provides wizards that enable you to create web services from a published business service. The JDeveloper tool includes the integrated WebLogic server that enables you to test your web service. After you create a web service, you deploy it to the integrated WebLogic server so that you can test the web service.

> **Note:** If you are using JD Edwards EnterpriseOne Tools Release 9.2 with JD Edwards EnterpriseOne Applications Release 9.2, JDeveloper 12c is installed on your system. With JDeveloper 12c, you can create only JAX-WS business services. If you are using JD Edwards EnterpriseOne Tools Release 9.2 with JD Edwards EnterpriseOne Applications Release 9.0 or 9.1, JDeveloper 11g is installed on your system. With JDeveloper 11g, you can create JAX-WS and JAX-RPC business services.

Creating a JAX-WS Web Service in JDeveloper

> **Note:** This section is applicable for JDeveloper 12c and JDeveloper 11g.

After you create and successfully compile a published business service, you can use JDeveloper to create a JAX-WS web service. You create the published business service manager and value object Java files using existing business service development methodology.

Before you run the JDeveloper JAX-WS Web Service wizard to create a web service from a published business service, you move the JAX-WS Web Services library to the top of the Libraries and Classpath properties page so that JDeveloper builds the published business service using the desired web-service standard.

> **Note:** The steps to create a JAX-WS web service also apply to creating a Media Object web service, except that you need to select the Enable MTOM check box in the Message Format page. See the steps in “To create a JAX-WS-based web service from a published business service" for details.

To select JAX-WS as the standard for a published business service:

1. In JDeveloper, right-click the appropriate published business service project and select Project Properties.

   The Project Properties window for the selected published business service appears.
2. In the left-hand side tree structure, select Libraries and Classpath.

   The Libraries and Classpath page appears on the right-hand side of the window.
3. In the Classpath Entries area, select JAX-WS Web Services.
4. Move the JAX-WS Web Services entry to the top of the page by clicking the Move Up button.

   Clicking the Move Up button moves the entry one level at a time; you may need to click the Move Up button multiple times to move the JAX-WS Web Services entry to the top of the page.
5. Click OK.
6. On JDeveloper, save the changes that you made to the Libraries and Classpath property of the selected published business service.

To create a JAX-WS-based web service from a published business service:

1. In JDeveloper, expand the appropriate published business service project folder.
2. Right-click on the published business service Manager Java file and select Create Web Service.

   The Create Java Web Service window appears with the Generation Options page available.
3. On Generation Options, verify the system entered values for the Web Service Name and the Port Name fields.
4. Click Next to open the Message Format page.
5. On Message format, do the following:
   - Select SOAP 1.1 Binding option.
   - Select Document/.Literal from the SOAP Message Format drop-down list.
   - **Note:** If you are creating a Media Object published business service, select the Enable MTOM check box. For more information about Media Object published business services, see *Understanding MTOM-Based Media Object Web Services* in this guide.
   - Click Next to open the Methods page.
6. On Methods, verify that all of the public methods that are defined for the published business service Manager Java file are listed and selected.
7. Click Next to open the Additional Classes page.
8. On Additional Classes, click Next to open the Configure Policies page.
9. On Configure Policies, select the No Policies option.
10. Click Next to open the Provide Handler Details page
11. On Provide Handler Details, verify the system entered Port and click Next to open the Finish page.
12. On Finish, click the Finish button to create the web service.

When JDeveloper successfully generates the JAX-WS web service, it also creates a web.xml file in the public_html/web-inf folder of the published business service project. JDeveloper also adds the following annotations to the beginning of the Java class in the PBSSV_Manager.java file.

- @SOAPBinding(parameterStyle = SOAPBinding.ParameterStyle.BARE)
- @WebService

If you plan to create a JAX-WS based client for this JAX-WS web service and you want the same client to work with both the EnterpriseOne JAX-WS web service running locally on the JDeveloper Integrated WebLogic server and on a standalone WebLogic Server or WebSphere Application Server or both, replace the above annotations that JDeveloper inserted at the beginning of the PBSSV_Manager Java Class with the following annotations:

- @SOAPBinding(style=SOAPBinding.Style.DOCUMENT, use=SOAPBinding.Use.LITERAL, parameterStyle=SOAPBinding.ParameterStyle.BARE)
- @WebService(serviceName="RI_AddressBookManagerService", portName="RI_AddressBookManagerPort", targetNamespace="http://oracle.e1.bssv.JPR01000/")

Note: The attributes in the @WebService annotation are specific for the RI_AddressBookManager published business service in the JPR01000 business service project. Change the attributes to reference the published business service that you specified for creating the JAX-WS web service.

Manual replacement of the annotations is required to set the *targetNamespace* attribute in the @WebService annotation with the proper package structure for the published business service project. If the replacement annotations are not specified, JDeveloper uses the reverse package structure; for example, JPR01000.bssv.e1.oracle, and in this scenario the client created for the local web service will not work with the server web service. If the above replacement annotations are made in the PBSSV_Manager Java Class, then they will be similar to the annotations added to the PBSSV_Manager Java Class during the business service package build process. A business service client created for a

published business service running locally on either the JDeveloper 11g or JDeveloper 12c integrated WebLogic server also works with a published business service running on a standalone WebLogic server.

Creating a JAX-RPC Web Service in JDeveloper

> **Note:** This section is applicable for JDeveloper 11g only. With JDeveloper 12c, development of JAX-RPC web services is not supported.

After you create and successfully compile a published business service, you can use JDeveloper 11g to create a JAX-RPC based web service. The following steps guide you through the process.

To select JAX-RPC as the standard for a published business service:

1. In JDeveloper, right-click on the appropriate published business service project and select Project Properties.
   The Project Properties window for the selected published business service appears.
2. In the left-hand side tree structure, select Libraries and Classpath.
   The Libraries and Classpath page appears on the right-hand side of the window.
3. In the Classpath Entries area, select JAX-RPC Web Services and JAX-RPC Web Logic Client.
4. Move both JAX-RPC 11 Web Services and JAX-RPC Web Logic Client entries to the top of the page by clicking the Move Up button.
   Clicking the Move Up button moves the entry one level at a time; you may need to click the Move Up button multiple times to move the entries to the top of the page.
5. Click OK.
6. On JDeveloper, save the changes that you made to the Libraries and Classpath property of the selected published business service.

To create a JAX-RPC-based web service from a published business service:

1. In JDeveloper, expand the appropriate published business service project folder.
2. Right-click on the published business service Manager Java file and select Create Web Service.
   The Create Java Web Service window appears with the Generation Options page available.
3. On Generation Options, verify the system entered values for the Web Service Name and the Port Name fields.
4. Click Next to open the Service EndPoint Interface page.
5. On the Service Endpoint Interface page, select the Autogenerate Service Endpoint Interface option, and verify the system entered value for this option.
6. Click Next to open the Message Format page.
7. On Message format, do the following:
   - Ensure the Generate Schema with Qualified elements option is selected.
   - Select Document/Literal from the SOAP Message Format drop-down list.
8. Click Next to open the Mapping page, and accept the system entered values for the fields.
9. Click Next to open the Methods page.
10. On Methods, verify that all of the public methods that are defined for the published business service Manager Java file are listed and selected.
11. Click Next to open the Choose Service State page.
12. On Choose Service State, click Next to open the Additional Classes page.
13. On Additional Classes, click Next to open the Select WLS policies page.
14. On Select WLS policies, click Next to open the Provide Handler Details page.

15. On Provide Handler Details, verify the system entered Port and click Next to open the Finish page.
16. On Finish, click the Finish button to create the web service.

Testing a Published Business Service Project on the JDeveloper Integrated WebLogic Server

If you used JDeveloper 12c to create a JAX-WS web service for a published business service, you deploy the published business service project to the JDeveloper 12c integrated WebLogic server to test the web service.

If you used JDeveloper 11g to create a JAX-RPC or JAX-WS web service for a published business service, you deploy the published business service project to the JDeveloper 11g integrated WebLogic server to test the web service.

Creating a Deployment Profile for the Published Business Service Project

Before you deploy the published business service project to the Integrated WebLogic server, you must create a deployment profile. You use JDeveloper to create a deployment profile for the published business service project.

To create a deployment profile:

1. On JDeveloper, right-click on the appropriate published business service project and select Project Properties.
2. On Project Properties, select the published business service project in the Development Profiles area, and then click Edit.
3. On Edit WAR Deployment Profile Properties, in the tree structure, open File Groups, then open WEB-INF/classes, and then select Contributors.
4. In the Contributors pane, select these options:
   - Project Output Directory
   - Project Dependencies

     In the Order of Contributors area, ensure that the EnterpriseOne install path (E1_Install_Path\PathCode\ini\sbf) appears, as shown in this example:

     ```
     C:\e900\DV900\ini\sbf
     ```
5. In the tree structure WEB-INF/classes file group, select Filters.
6. In the Filters pane, select the Files tab and verify the following
   - All preconfigured EnterpriseOne ini files are selected.
   - In the oracle folder, the published business service and all business service projects on which the published business service is dependent are selected.
7. In the tree structure, open the WEB-INF/lib file group, and select Contributors.
8. On the Contributors page, select SBFProjects in the Libraries Selected for Deployment area.
9. In the tree structure WEB/INF/lib file group, select Filters.
10. In the Filters page, select the Patterns tab and ensure that all of the EnterpriseOne tools foundation JAR files that are required for the business service are selected.
11. In the tree structure, select the Library Dependencies group file.
12. In the Library Dependencies page, ensure that all of the libraries under the Libraries Selected for Deployment area are selected.

13. In the tree structure, open the Platform group file.
14. In the Platform page, select IntegratedWebLogicServer from the Target Connection drop-down list.
15. Click OK on the Edit WAR Deployment Profile properties window.
16. On the Deployment window, click OK.
17. On JDeveloper, select the published business services project and click Save on the File menu.

Deploying the Published Business Service Project to the Integrated WebLogic Server

After you create the deployment profile, deploy the published business service project to the integrated WebLogic server.

To deploy the published business service project to the Integrated WebLogic server:

1. Start the Integrated WebLogic server.
2. On JDeveloper, right-click on the appropriate published business service project, select Deploy, and then select <published business service project> to deploy to IntegratedWebLogicServer.
3. Ensure deployment of the published business service project is successful by verifying the Deployment Finished message that appears in the Deployment-Log window in the Deployment tab in JDeveloper.

Testing the Deployed Published Business Service

After you successfully deploy the JAX-RPC or JAX-WS-based published business service to the integrated WebLogic server, you test it. You can use one of these two URLs to access the WebLogic Server Administration Console:

- `http://localhost:7101/console/login/Loginform.jsp`
- `http://IPorMachineNameofJdev:7101/console/login/Loginform.jsp`

To test the JAX-RPC or JAX-WS-based published business service on the Integrated WebLogic server:

1. Access and log in to the Integrated WebLogic Server Admin Console.
2. In left-hand pane on the Integrated WebLogic Server Admin console, in the Domain Structure area, open the Environment domain and then select Servers, Default Server, Deployments.

   The Settings for Default Server page appears on the right-hand side of the admin console.
3. Select the Deployments tab.
4. In the Applications and Modules Deployed for this Server area, expand the appropriate published business service.
5. Expand the Web Services name, and click on your web service; an example web service name is RI_AddressBookManagerService.

   The Settings for the named web service page appears on the right-hand side of the admin console.
6. Select the Testing tab.

   The Deployment Tests page appears on the right-hand side of the admin console.
7. In the Deployments Tests area, click the Test_client link in the Test Point column.

   On the WebLogic Test client tab, the operations for the web service appears.
8. Depending on the operation that you want to invoke, modify the request document and test the business service.

For example to invoke the getAddressBook operation of the RI_AddressBookManager web service, modify the request document as follows, and then click the getAddressBook button.

```
<getAddressBook xmlns="http://oracle.e1.bssv.JPR01000/">
    <entity xmlns="">
        <entityId>4242</entityId>
    </entity>
</getAddressBook>
```

9. On the Service Response, verify the test was successful.

When testing the JAX-RPC or JAX-WS published business service, you can configure these logs to collect information for troubleshooting issues that might occur.

- Business Services logs

  In the jdelog.properties file, located in the <E1_Install_Path>\ini\sbfFolder, turn on the BSSV log with Level=Debug and Component=ALL. If you change the jdelog.properties file, you must redeploy the published business service project to the Integrated WebLogic server.

- IntegratedWeblogicServer log

  For JDeveloper 12c, this log is located at:

  ```
  C:\Documents and Settings\USER_PROFILE\Application Data
  \JDeveloper\system12.1.2.*\DefaultDomain\servers\DefaultServer\logs
  ```

  For JDeveloper 11g, this log is located at:

  ```
  C:\Documents and Settings\USER_PROFILE\Application Data \JDeveloper\system11.1.1.*
  \DefaultDomain\servers\DefaultServer\logs
  ```

Tip: If your target web service is behind OHS 12.1.3, then add the following setting to your SOAPUI.

```
C:\Documents and Settings\USER_PROFILE\Application Data\JDeveloper\system11.1.1.*
\DefaultDomain\servers\DefaultServer\logs
```

Understanding MTOM-Based Media Object Web Services

With Java API for XML-Based Web Services (JAX-WS), you can send binary attachments such as images or files along with web services requests. JAX-WS adds support for optimized transmission of binary data as specified by SOAP MTOM. MTOM is the W3C Message Transmission Optimization Mechanism, a method for sending binary data to and from web services. Media Object published business services leverage the MTOM specification to transmit media objects as binary data in SOAP messages.

Creating a Media Object Web Service in JDeveloper

The steps to create a Media Object web service are the same as creating a JAX-WS web service in JDeveloper. However, you need to select the **Enable MTOM** check box in the Message Format page. See *Creating a JAX-WS Web Service in JDeveloper* in this guide for detailed steps.

Both JDeveloper 11g and JDeveloper 12c support creating media object web services.

Testing the Deployed Media Object Published Business Service

This section provides a brief overview of how you can test the Select, Insert, and Delete Media Object operations of a Media Object published business service.

Select

To test the Select operation, you should test retrieving a single media object and also test retrieving multiple media objects. If you want to retrieve a single record at a time, you must provide the sequence number of the media object through the value object. If you do not provide the sequence number, then the Select operation may retrieve multiple media objects. You can select the media objects based on the Media Object type. In this case, you will retrieve all of the media objects of that Media Object type for that key (such as address book number in the case of RI_AddressBookMediaObjectManager BSSV). If you do not provide the sequence number and Media Object type, then the select operation will retrieve all of the media objects for that key.

The following sample code is an XML snippet for selecting multiple media objects based on the Media Object type FILE. This will return all media objects for address book number 1001 with a Media Object type of FILE.

```
<mediaObject>
            <mnAddressNumber>1001</mnAddressNumber>
            <moItems>
               <szMoType>FILE</szMoType>
            </moItems>
 </mediaObject>
```

The following sample code is an XML snippet for selecting a single attachment based on the sequence number. This will return a media object with a sequence number of 1 for address book number 1001.

```
<mediaObject>
            <mnAddressNumber>1001</mnAddressNumber>
            <moItems>
               <moSeqNo>1</moSeqNo>
            </moItems>
 </mediaObject>
```

The following sample code is an XML snippet for selecting all the attachments. This will return all media objects, including media objects of type FILE, URL and TEXT, for address book number 1001.

```
<mediaObject>
            <mnAddressNumber>1001</mnAddressNumber>
            <moItems>
            </moItems>
 </mediaObject>
```

In addition, there is an option to list only the media object details such as sequence number, media object name, media object type and so forth, without the attachments. In this case, you would set the downloadMediaObject member

variable to `false` in the internal Media Object value object. The preceding sample XML inputs for a Select operation are valid for listing media objects.

Insert

To test the Insert operation of any media object, you have to test inserting the different Media Object types supported, which are FILE, TEXT and URL.

If you are inserting a FILE attachment, provide the following information:

- Media Object type (<szMoType>) as FILE.
- Attachment data (<szData>).
- Media Object Item Name (<szItemName>). Enter the file name of the media object in this field.

If you are inserting the TEXT attachment, provide the following information:

- Media Object type (<szMoType>) as TEXT
- Attachment data (<szData>).

  Media Object Item Name is not required for TEXT media objects.

If you are inserting a URL attachment, provide the following information:

Media Object type (<szMoType>) as URL.

Media Object Item Name (<szItemName>). Enter the URL to be stored in this field.

The following sample code is an XML snippet for inserting three media objects: a FILE attachment, a URL attachment, and a TEXT attachment:

```
<mediaObject>
             <moItems>
                <szData>cid:README.pdf</szData>
                <szItemName>README.pdf</szItemName>
                <szMoType>FILE</szMoType>
             </moItems>
        <moItems>
                <szData>cid:text.txt</szData>
                <szMoType>TEXT</szMoType>
             </moItems>
             <moItems>
                <szItemName>www.oracle.com</szItemName>
                <szMoType>URL</szMoType>
             </moItems>
</mediaObject>
```

Delete

To test the Delete operation, test deleting a single media object and also test deleting multiple media objects in a single web service call. If you want to delete only a single media object at a time, provide the sequence number of the media object through the value object. If you do not provide the sequence number, then the Delete operation may delete multiple media objects based on the query criteria.

You can delete media objects based on the Media Object type. When based on the Media Object type, all media objects of that particular Media Object type for that key (such as address book number in the case of RI_AddressBookMediaObjectManager BSSV) will be deleted. If you do not provide both the Media Object sequence number and Media Object type, then the Delete operation will delete all the media objects for that key.

The following sample code is an XML snippet for deleting multiple media objects based on the Media Object type FILE. This will delete all FILE type media objects for address book number 1001.

```
<mediaObject>
            <mnAddressNumber>1001</mnAddressNumber>
            <moItems>
               <szMoType>FILE</szMoType>
            </moItems>
 </mediaObject>
```

The following sample code is an XML snippet for deleting a single attachment based on the sequence number. This will delete the media object with a sequence number of 1 for address book number 1001.

```
<mediaObject>
            <mnAddressNumber>1001</mnAddressNumber>
            <moItems>
               <moSeqNo>1</moSeqNo>
            </moItems>
 </mediaObject>
```

The following sample code is an XML snippet for deleting all attachments. This will delete all media objects, including media objects of type FILE, URL and TEXT, for address book number 1001.

```
<mediaObject>
            <mnAddressNumber>1001</mnAddressNumber>
            <moItems>
            </moItems>
 </mediaObject>
```

Before you test a JAX-WS-based Media Object published business service, you follow the same preliminary steps that you would perform to test any published business service, which include:

- Creating a deployment profile for a Media Object published business service.
- Deploying the Media Object published business service project to the integrated WebLogic Server.

See *Testing the Deployed Published Business Service* in this guide for details.

After you successfully deploy the Media Object published business service to the integrated WebLogic Server, test it. You can use the following URL to access the WebLogic Server Administration Console:

http://<JDEV_MACHINE_HOST>:<JDEV_INTEGRATED_WLS_SERVER_PORT>/console/login/Loginform.jsp

To test the JAX-WS-based Media Object published business service on the integrated WebLogic Server:

1. Access and log in to the integrated WebLogic Server Administration Console.
2. In the "Applications and Modules Deployed for this Server" area, expand the appropriate published business service.
3. Expand the Web Services name and click your web service; an example web service name is **RI_AddressBookMediaObjectManagerService.**

   The "Settings for web service name" page is displayed in the Administration Console.
4. Select the Testing tab to access the Deployment Tests page.
5. In the Deployments Tests area, expand the name and click on the ?WSDL in the Test Point column.

   WSDL is displayed on the new tab. Record or note the WSDL URL.
6. For testing business services, open a web service testing tool such as SOAPUI and select File, New SOAPUI Project.
7. Enter the WSDL URL, which you recorded in step 5, enter the Project name, and then click OK.

8. The SoapUI project appears in the left pane. Expand your Soap Operation and double-click Request1.

   Depending on the operation that you want to invoke, modify your request document and test your business service.

   Example: The following example shows the input xml and configuration steps for invoking the addAddressBookMO operation of the RI_AddressBookMediaObjectManager web service, which will insert two FILE media objects and one TEXT media object.

```
<orac:addAddressBookMO>
        <businessUnit>1</businessUnit>
        <entityName>Media Object Test</entityName>
        <entityTypeCode>C</entityTypeCode>
        <mediaObject>
            <moItems>
               <szData>cid:README.pdf</szData>
               <szItemName>README.pdf</szItemName>
               <szMoType>FILE</szMoType>
            </moItems>
    <moItems>
               <szData>cid:text.txt</szData>
               <szMoType>TEXT</szMoType>
            </moItems>
            <moItems>
               <szData>cid:Green.gif</szData>
               <szItemName>Green.gif</szItemName>
               <szMoType>FILE</szMoType>
            </moItems>
        </mediaObject>
</orac:addAddressBookMO>
```

   a. Click the Add an Attachment button and browse and select for your attachment. You may need to repeat this step to add multiple attachments.
   b. Change the request parameters to **`Enable MTOM = true`** and **`Disable Multiparts = False`**.

      These properties are displayed in the lower left side of the SOAPUI pane.
   c. Submit the request.
   d. Check the Service Response to verify the invocation was successful.

   Example: The following example shows the input xml and configuration steps for invoking the getAddressBookMO operation of the RI_AddressBookMediaObjectManager web service, which will select all media objects of type FILE for address book number 605864.

```
      <orac:getAddressBookMO>
        <entity>
            <entityId>605864</entityId>
         </entity>
         <mediaObject>
            <mnAddressNumber>605864</mnAddressNumber>
            <moItems>
               <szMoType>FILE</szMoType>
            </moItems>
 </mediaObject>
       </orac:getAddressBookMO>
```

   a. Change the request parameters to "Enable MTOM = true" and "Disable Multiparts = False"

      These properties are displayed in the lower left side of the SOAPUI pane.

b. Submit the request.
c. Check the Service Response to verify the invocation was successful.
d. If you choose to do so, you can save the Attachments by clicking the Export Icon in the Attachment Pane of the response.
e. On the Service Response, verify the test was successful.

Example: The following example shows the input xml and configuration steps for invoking the deleteAddressBookMO operation of the RI_AddressBookMediaObjectManager web service, which will delete all media objects of type FILE for address book number 605864.

```
<orac:deleteAddressBookMO>
        <entity>
            <entityId>605864</entityId>
         </entity>
         <mediaObject>
            <mnAddressNumber>605864</mnAddressNumber>
            <moItems>
               <szMoType>FILE</szMoType>
            </moItems>
 </mediaObject>
      </orac:deleteAddressBookMO>
```

a. Submit the request.
b. Check the Service Response to verify the invocation was successful.

When testing the business service, you can configure the following logs to collect information for troubleshooting issues that might occur:

- Business services logs

  In the jdelog.properties file, located in the **<E1_Install_Path>\ini\sbfFolder**, turn on the business service log with **Level=Debug** and **Component=ALL**. If you change the jdelog.properties file, you must redeploy the published business service project to the integrated WebLogic server.

- IntegratedWeblogicServer log

  For JDeveloper 12c, this log is located at:

  ```
  C:\Documents and Settings\USER_PROFILE\Application Data
  \JDeveloper\system12.1.2.*\DefaultDomain\servers\DefaultServer\logs
  ```

  For JDeveloper 11g, this log is located at:

  ```
  C:\Documents and Settings\USER_PROFILE\Application Data \JDeveloper\system11.1.1.*
  \DefaultDomain\servers\DefaultServer\logs
  ```

7 Working with Business Service Properties

Understanding Business Service Properties

Business service properties are used with JD Edwards EnterpriseOne business services and provide a way for you to change a value in the business service without changing the business service code. A business service property consists of a property key and a property value. The property key is the name of the business service property and cannot be changed. The property value is the value that you provide for the property key. You can modify the property value. You can include a business service property method in the published business service.

Business service properties are unique to an environment. For example, if you change a property value in your test environment, the business service property that is in your production environment will have the original property value unless you change it, too.

Naming conventions and methodology for using business service properties are discussed in the Business Services Development Methodology guide.

See *"System-Level Business Service Properties" in the JD Edwards EnterpriseOne Tools Business Services Development Methodology Guide* .

Business Service Property Utility Classes

These utility classes are available for the business service property:

- ServicePropertyAccess
- ServicePropertyException

You use ServicePropertyAccess to access property values.

You use ServicePropertyException to find and handle errors when accessing business service properties.

You use these two APIs to retrieve a business service property:

- Public static String getSvcPropertyValue(Context context, String key) throws ServicePropertyException.
  - Retrieves the property value for a given key.
  - Requires context to obtain connection to the JD Edwards EnterpriseOne database.
  - Returns the value stored in the database.
  - Throws ServicePropertyException if an error occurs.
- Public static String getSvcPropertyValue(Context context, String key, String defaultVal) throws ServicePropertyException.
  - Retrieves the property value for a given key.
  - Requires context to obtain connection to the JD Edwards EnterpriseOne database.
  - Returns the value stored in the database if it is not null or blank.
  - Returns the passed default value if the value in the database is null or blank.
  - Throws ServicePropertyException if an error occurs.

Errors and Error Logging

If an error occurs during retrieval of a business service property, the system rolls back any changes, logs the error, and throws an exception. Error message handling is managed by the business service method. You can design your business service method so that when the system throws an exception, an error message is sent to the caller.

The following list identifies possible business service property errors that throw an exception and cause an error message to be returned to the caller:

- Security credentials do not have authority to read business service properties.
- The specified property key does not exist.
- The value for the property key is null, and no default value is provided.
- The values for the business service property record are incomplete.

Managing Business Service Properties

This section provides an overview of managing business service properties and discusses how to:

- Add a business service property record.
- Modify a business service property record.

Understanding Business Service Property Information

You can use the Business Service Property Program (P951000) to manage business service property information. With this program, you can add or delete business service properties, and you can modify the property value. You can also use Object Management Workbench (OMW) to create or delete a business service property or to modify a property value. If you need to create several business service properties, consider using P951000 to create the business service property and then add each business service property to the OMW project. The benefit of using OMW to create a business service property is that the object is automatically put into your OMW project for you.

When you add a new business service property, you indicate whether it is a system-level or service-level business service property. Business service properties categorized at the system level are used by more than one business service. Business service properties categorized at the business service level are used by only one business service.

After you create a business service property, you cannot change the name, because this is the key that the business service uses to call the business service property. You can change the property value.

All business service properties are stored in the Business Service Property table (F951000). You can view system-level, business service-level, or all business service properties that are available in your login environment from the Work with the Business Service Properties form.

Note:

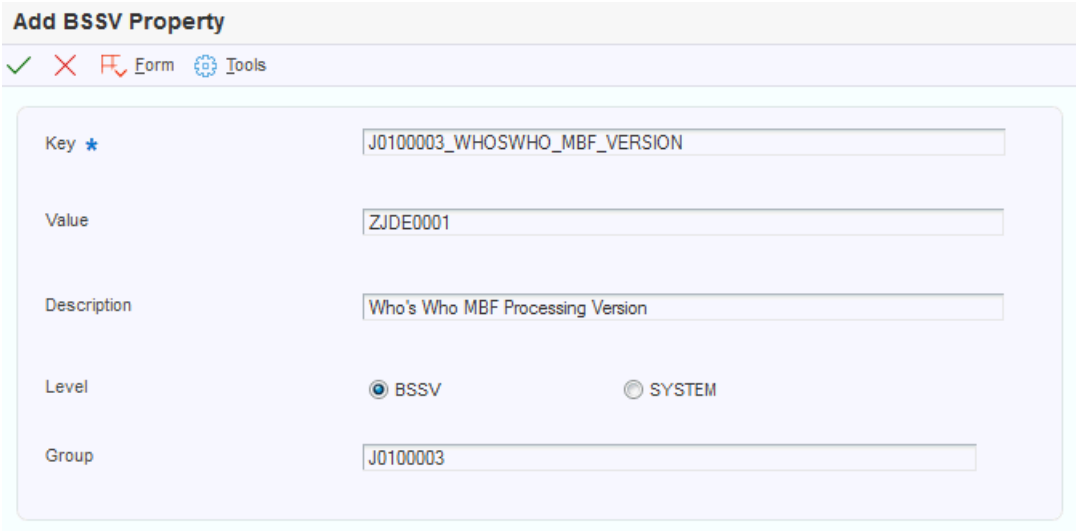
- *"Adding a Business Service Property" in the JD Edwards EnterpriseOne Tools Object Management Workbench Guide* .

Forms Used to Manage Business Service Properties

Form Name	FormID	Navigation	Usage
Working with Business Service Properties	W951000F	From the System Administration Tools menu, select Business Service Property from the Business Service Property and Business Service Cross Reference Administration folder.	Find, modify, or delete business service properties.
Add BSSV Property	W951000C	From Work with Business Service Properties, click Add.	Create a new business service property or modify the value of an existing business service property.

Adding a Business Service Property Record

Access the Add BSSV Property form.



Key
A name that uniquely identifies the business service property. This name cannot be changed. The length of the property key can be up to 255 characters.

Value
An entry that defines specific criteria. When you process the business service, you can change the value of the service property key. For example, the original value might be ZJDE0001 but when you process the business service, you might want to change this value to ZJDE0002.

Description
A phrase or sentence that identifies the purpose of the business service property.

Level
Select a way to group business service properties for viewing. Values are:

- **BSSV:** Shows all business service properties for a specific business service.
- **SYSTEM:** Shows all business service properties that you have defined as system-level business service properties. These business service properties can be used by more than one business service.
- **AIS (Release 9.2.6.4):** This selection enables you to access the property from the Orchestrator Studio.
- **All:** Shows all business service properties that you have defined in your login environment.

Group
A way to classify the business service property at the business service level. The group name must be an existing business service name.

Modifying a Business Service Property Record

Access the Work with Business Service Properties form, select the business service property to be changed, and then click Select.

On the Modify Business Service Property form, you can change the Value, Description, Level, and Group fields. You cannot change the Key field.

8 Working with JD Edwards EnterpriseOne as a Web Service Consumer

Working with JD Edwards EnterpriseOne as a Web Service Consumer

Note: Oracle reserves the right to reorganize the business services foundation packages (jar files) for tools release upgrades. If you are planning to upgrade your system, test your custom objects and modify them as appropriate to ensure your code will continue to work as intended. You cannot upgrade custom business service objects after you install a tools release upgrade.

Understanding JD Edwards EnterpriseOne as a Web Service Consumer

JD Edwards EnterpriseOne can call and process external web services. Being a native consumer of web services enables JD Edwards EnterpriseOne to integrate with other Oracle products and third-party systems and products, such as order promising. To enable JD Edwards EnterpriseOne as a web services consumer, you create a business service that uses a web service proxy to call an external web service. A method in that business service can be called by a JD Edwards EnterpriseOne business function. The business service resides on the business services server and is written in the Java programming language. The business service contains a web service proxy that you generate using JDeveloper. The business function resides in your enterprise server and is written in the C programming language. The business function calls the business service method.

Communicating Between JD Edwards EnterpriseOne Servers

The business function, which resides in your enterprise server, uses an API to call the business service method, which resides on the business services server. JDENet is the communication mechanism between the servers.

You can configure your system to use either one port or multiple ports to listen for and receive JDENet messages from the Enterprise Server. For single port communication, an Object Configuration Manager (OCM) mapping tells the enterprise server the business services server name and port for sending the JDENet messages.

See *Setting Up OCM for Business Functions Calling Business Services*.

For multiple port communication, you use a clustered business services server to listen for and receive multiple JDENet messages from the Enterprise Server.

Business Function APIs for Calling a Business Service

These APIs are available for a business function to call a business service method:

- jdeCallBusinessService
- jdeCallBusinessServiceNoResponse
- jdeFreeBSSVPayloadReturn
- jdeGetBusinessServiceErrorText

Two APIs are available for calling a business service method from a business function:

- jdeCallBusinessService

  This API makes a synchronous call to the business services server passing in an XML document that represents the value object for the called business service method. The call waits until the business service method finishes before processing.

- jdeCallBusinessServiceNoResponse

  This API makes an asynchronous call to the business services server passing in an XML document that represents the value object for the called business service method. The API returns immediately after sending the message without waiting for a response.

For both of these APIs, the caller must allocate the bssvPayloadInput structure. In addition, the caller must free the bssvPayloadReturn structure with the provided API.

This example code provides a typical C business function call to a business service method:

```
* jdeCallBusinessService (lpBhvrCom,
*               lpVoid,
*               _J("oracle.e1.sbf.JXXXXX")
*               _J("bssv_method_name_to_call")
*               TRUE,
*               bssvPayload,
*               &bssvPayloadReturn);
*
* /* caller must free the space allocated for the return payload when
*    finished with it */
* jdeFreeBSSVPayloadReturn (&bssvPayloadReturn);
*
```

Both APIs take the same set of parameters, which is shown in this example code:

```
* lpBhvrCom - BSFN structure pointer.
* lpVoidInfo - holds error info returned from BSFN.
* bssvName - fully qualified name of business service to be called
* bssvMethod - method within business service to be called
* autocommitTransaction - type of transaction the business service should use
*                         on the business services server.
*                         True if auto commit - False for manual commit.
* bssvPayloadInput - string representation of an XML document that represents
*                    the value object for the business service. Populated
*                    with input values to pass to the business service.
* bssvPayloadReturn - string representation of XML document representing the
*                     value object returned from the business service call
*                     populated with both return values and original input
*                     values. The caller must free the space allocated for
*                     the return payload by calling the
```

```
*                    jdeFreeBSSVPayloadReturn API.
```

Both APIs have the same return values, which are shown in this example code:

```
* CallBSSVNoError                     100L
* CallBSSVServiceDoesNotExist         101L
* CallBSSVMethodDoesNotExist          102L
* CallBSSVInvalidMethodSignature      103L
* CallBSSVNoDefaultConstructor        104L
* CallBSSVErrorUnMarshal              105L
* CallBSSVErrorMarshal                106L
* CallBSSVConnectionException         107L
* CallBSSVInvalidPacketNum            108L
* CallBSSVInvaldUserName              109L
* CallBSSVInvalidEnv                  110L
* CallBSSVInvalidRole                 111L
* CallBSSVInvalidToken                112L
* CallBSSVAuthenticationFailure       113L
* CallBSSVNoErrorWithMessages         114L
* CallBSSVInvalidParamError           201L
* CallBSSVSystemConfigurationError    202L
* CallBSSVSystemNetworkFailedError    203L
* CallBSSVSystemInternalServerError   204L
```

Note:

- *JD Edwards EnterpriseOne Tools API Reference Guide* on the My Oracle Support Web site.

Xerces APIs

You can use any one or more Xerces APIs in combination with other XML generation and editing APIs to generate the XML payload for calls to a business service. You can use any of these Xerces APIs, which have been added to the Xerces wrapper:

- Xerces APIs for working with MathNumeric data types:
  - XRCS_createElementFromMathNumeric
  - XRCS_setMathNumericElementValue
  - XRCS_getMathNumericFromElement
- Xerces APIs for working with JDEDATE data types:
  - XRCS_createElementFromJDEDate
  - XRCS_setJDEDateElementValue
  - XRCS_getJDEDateFromElement
- Xerces APIs for working with JDEUTIME data types:
  - XRCS_createElementFromJDEUTIME
  - XRCS_setJDEUTIMEElementValue
  - XRCS_getJDEUTIMEFromElement
- Xerces APIs for parsing/serialization ignoring encoding tags:

- XRCS_serializeDocumentToXMLStringNoEncoding
- XRCS_parseXMLStringRemoveEncoding

Note:

- *JD Edwards EnterpriseOne Tools API Reference Guide* on the My Oracle Support Web site.

Setting Up OCM for Business Functions Calling Business Services

This section provides an overview of setting up OCM for business functions calling business services and discusses how to:

- Configure OCM for business functions calling business services.
- Ping the business services server.

Understanding OCM Setup for Business Functions Calling Business Services

Typically, a system administrator configures OCM so that your enterprise server can find the business services server. After OCM is configured with the business services server information, you can ping the business services server from OCM to ensure that the business services server is running and listening for messages.

Note:

- *"Understanding and Working with Object Configuration Manager" in the JD Edwards EnterpriseOne Tools System Administration Guide*

Forms Used to Set Up OCM for Business Functions Calling Business Services

Form Name	FormID	Navigation	Usage
Machine Search and Select	W986110D	Expand the System Administration Tools folder on the EnterpriseOne Life Cycle Tools / All My Roles menu, and then select Environment / Service Configuration from the Environment Management folder.	Select the appropriate machine name and data source combination.

Form Name	FormID	Navigation	Usage
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 business services server and server port, or access the Service Configuration Revisions form to add a new configuration record. Also ping the business services server to determine whether the business services server is available.
Service Configuration Revisions	W986110K	On Work with Service Configurations, click Add.	Add a new configuration in OCM with the location of the business services server.

Configuring OCM for Business Functions Calling Business Services

Access the Service Configuration Revisions form and complete the following fields:

Environment Name
A name that uniquely identifies the environment.

Service Name
A name that identifies the type of server. For example, BSSV identifies the business services server.

User/Role
A specific JD Edwards EnterpriseOne user name, role name, or *PUBLIC to whom the record applies.

Server
The name of the local host where the business services server is running.

Port
The port number of the business services server. This is the JDENet listening port.

Pinging the Business Services Server

Access the Work with Service Configurations form.

1. In the grid area, highlight the server.
2. From the Row menu, select Change Status to activate the configuration.
3. From the Row menu, select Ping Server.

   You receive a Ping message indicating whether the Ping test was successful or failed.

Developing a Business Service for Consuming an External Web Service

This section provides an overview of how to develop a business service for consuming an external web service and discusses how to:

- Identify an external web service.
- Create a business service object.
- Create a web service proxy prior to JDeveloper 11g.
- Create a web service proxy for a web service deployed in WebLogic Server using JDeveloper 11g with JAX-RPC.
- Create a value object class.
- Create a business service class.
- Generate a sample XML document.

Understanding How to Develop a Business Service for Consuming an External Web Service

You create a business service that consumes an external web service in the same way that you create a business service that provides a web service for public consumption. But instead of using a published business service as the web service, the business service is used to consume the web service. As part of the process, you create a web service proxy and an XML document. This business service is called by a business function. The following tasks provide a scenario for creating a JD Edwards EnterpriseOne business service that consumes an external web service.

> **Note:** If you are using JD Edwards EnterpriseOne Tools Release 9.2 with JD Edwards EnterpriseOne Applications Release 9.2, JDeveloper 12c is installed on your system. With JDeveloper 12c, you can create only JAX-WS business services. If you are using JDeveloper 12c, do not use JDK 1.7 specific APIs in any of the published business services code. If you are using JD Edwards EnterpriseOne Tools Release 9.2 with JD Edwards Enterprise One Applications Release 9.0 or 91, JDeveloper 11g is installed on your system. With JDeveloper 11g, you can create JAX-WS and JAX-RPC business services.

Identifying an External Web Service

Before you create a business service to consume an external web service, you need to identify the external web service and its location. The external web service definition determines the data to be sent to the web service and the result that is received.

Creating a Business Service Object

Using Object Management Workbench (OMW), create the object that will contain the business service being developed. Start JDeveloper from OMW to work with this project.

Creating a Web Service Proxy

A web service proxy is a collection of classes generated by JDeveloper and is used to call external web services. JDeveloper uses the target WSDL to create all necessary classes. A JD Edwards EnterpriseOne business service method uses the web service proxy to call the external web service. You use JDeveloper to create a web service proxy in the business service project that you created using OMW.

When you create a new business service proxy project in JDeveloper to consume a web service running on the WebLogic Server (WLS), JDeveloper includes certain XML files in the proxy project. When you check in the business service proxy project through OMW, the XML files in the proxy project must also be checked in as part of the project.

To be able to check in XML files along with the business service consumer projects through OMW, ensure that the UDC H95/CA includes records for XML files as business services artifacts that can be checked in.

Ensure that any new business service proxy project created in JDeveloper to consume a web service is checked in through OMW with the JDeveloper install path specified as the JDeveloper install path.

After JDeveloper creates the web service proxy, you must rename the package folder, and then rebuild the code to ensure that no errors occurred.

For more information about creating a web service proxy, see one of the following:

- *Creating a JAX-WS-Based EnterpriseOne Consumer Business Service Using JDeveloper 12c or JDeveloper 11g*
- *Creating a Web Service Proxy Using JDeveloper11g with a JAX-RPC Client*

Creating a Value Object Class

After you create the web service proxy, you create the value object class. You create the value object class based on data objects in the JD Edwards EnterpriseOne business function that will be used to call this business service. Then, you must write the code to map between data types in the value object class and the data types required by the external web service.

Passing Data

You design your business service method so that it gets the data that will be passed to the web service from either the business services server or the enterprise server. When you design your business service method, you need to consider the amount of data being passed. The more data that is passed to the business service method, the larger the memory requirements are on the enterprise server. Consuming memory on the enterprise server can significantly impact other users of the enterprise server. Creating the value object that is passed to the business service method requires careful consideration. Here are some questions you might consider:

- Is it better to assemble the data from the business services server instead of the enterprise server?
- Can key information be passed instead of constructing the entire data set on the C (enterprise server) side?

- Is one approach clearly better than the other based on the requirements?

Depending on your business service design, data is either received in the business service value object, or the business service retrieves the data from a table or business function call. Either way, data type conversions occur within the business service method. Everything that is specific to the web service being called, including data type conversions, should be contained within the business service method that wraps the web service call.

Creating a Business Service Class

After you create the web service proxy and the value object class, you create a business service class. This business service is a wrapper for the web service proxy. The business service wrapper is necessary because:

- Only internal business services that match the JD Edwards EnterpriseOne business service methodology can be called from the enterprise server.
- Most likely some conversion between the JD Edwards EnterpriseOne data types and the data types required by the web service provider will be required.

To create a business service class:

1. Add classes to the business service object.
    - Create the business service class.
    - Name the business service class.
    - Create a transaction, if necessary (IConnection object).
    - Declare a business service public method.
2. Add business service logic:
    - Map from the value object to the format needed by the external web service.
    - Create a call to the web service proxy.
    - Use business service properties.
    - Use softcoding.
    - Handle errors.
    - Format data.

After you create the business service for calling an external web service, you can create a JD Edwards EnterpriseOne business function that calls this business service. You can also create or use an existing JD Edwards EnterpriseOne application to call the business function.

Generating a Sample XML Document

You use the XML Template utility to create a sample XML document. The sample XML document is based on the value object of the business service method that is calling an external web service. This XML document provides a model for creating the XML payload in the JD Edwards EnterpriseOne business function. The XML document represents the structure of the data that is sent from the business function. The XML Template utility is provided in JDeveloper, and you access the utility by selecting a value object Java file in the JDeveloper Application Navigator pane.

To invoke the XML Template utility that generates a sample XML document:

1. Start JDeveloper from OMW.
2. In the JDeveloper navigator pane, right-click the value object of the business service method that you want to call.
3. Select Generate XML Document Template from the context menu.

   The XML Template utility creates a new XML file in the same directory as the value object. The XML file has the same name as the value object, with an extension of XML instead of Java.
4. Click the refresh icon in the JDeveloper navigator pane.
5. Open the sample XML document by double-clicking the new XML file.

Testing a Business Service That Consumes an External Web Service

If you are using JDeveloper 12c to create a business service that consumes an external web service, you use the JDeveloper 12c integrated WebLogic Server to test the web service.

If you are using JDeveloper 11g or JDeveloper 12c to create a business service that consumes an external web service, you can use the JDeveloper integrated WebLogic Server to test the web service.

See the following sections in this guide to deploy and test the business service consumer project:

- *Creating a Deployment Profile for the Business Service Consumer Project*
- *Deploying the Business Service Consumer Project to the Integrated WebLogic Server*
- *Testing the Business Service Consumer Project*

You can test the business service using one of these methods:

- Create a test business service.
- Use the development business services server.

Guidelines for using these methods are provided in Appendix B: Testing a Business Service That Calls an External Web Service.

Note:

- *Creating a Test Business Service*.

Creating a JAX-WS-Based EnterpriseOne Consumer Business Service Using JDeveloper 12c or JDeveloper 11g

You use OMW to create a new business service object, and then open the business service object in JDeveloper from OMW. Before you create the business service, ensure that the target JAX-WS web service for which the JAX-WS-based EnterpriseOne client business service is being developed is up and running and its WSDL URL is accessible.

A JAX-WS based EnterpriseOne Consumer business service reference implementation, JRH90134, is provided for you to use as a model for creating a custom JAX-WS based consumer business service.

See *"Reference Implementation for Testing the Business Services Server as a JAX-WS Web Service Consumer" in the JD Edwards EnterpriseOne Tools Interoperability Reference Implementations Guide* .

Note: If you are using JD Edwards EnterpriseOne Tools Release 9.2 with JD Edwards Enterprise One Applications Release 9.0 or 9.1, and you plan to generate a web service based on the JAX-WS standard, it is important to consider your current consumption of the service. While the services function in the same manner, whether deployed as JAX-RPC or JAX-WS, there are some important caveats:

- The service wsdl for a service generated as JAX-WS differs from the same service wsdl generated as JAX-RPC.

  If you currently interact with these services from an external system you may need to make changes to those systems before switching to JAX-WS.
- For prepackaged integrations that use internal consumer services, you may need to maintain a JAX-RPC based deployment for them to function correctly.

  See the specific EnterpriseOne applications guide for your integration to view the supported BSSV deployment options.

See the technical paper, "Understanding the Impact of JAX-WS", on My Oracle Support (Doc ID 1488986.1) `https://support.oracle.com/rs?type=doc&id=1488986.1` for additional information about the implications of the JAX-WS build option.

Creating a JAX-WS Web Service Proxy in JDeveloper

You use the JDeveloper wizard to create a custom JAX-WS web service consumer.

To select JAX-WS as the standard for creating the JAX-WS based business service consumer project:

1. In JDeveloper, right-click on the appropriate business service consumer project and select Project Properties.

   The Project Properties window for the selected published business service appears.

2. In the left-hand side tree structure, select Libraries and Classpath.

   The Libraries and Classpath page appears on the right-hand side of the window.
3. In the Classpath Entries area, select JAX-WS Web Services.
4. Move the JAX-WS Web Services entry to the top of the page by clicking the Move Up button

   Clicking the Move Up button moves the entry one level at a time; you may need to click the Move Up button multiple times to move the JAX-WS Services entry to the top of the page.
5. Click OK.
6. On JDeveloper, save the changes that you made to the Libraries and Classpath property of the selected published business service.

To create a JAX-WS Web Service Proxy:

1. In JDeveloper, right-click on the appropriate business service client project and select New to open the New Gallery Window.
2. On the New Gallery window, select the All Technologies tab.
3. In the left-hand tree structure, expand the Business Tier category and select Web Services.
4. Select Web Service Proxy from the list of items that appears in the right-hand pane.
5. Click OK.

   This action opens the Create Web Service Proxy Wizard.
6. On the Welcome page click Next to open the Select Web Service Description page.

7. On Select Web Service Description, do the following:
   - In the WSDL Document URL field, enter the WSDL URL of the targeted JAX-WS based web service.

     The WSDL URL is that of the targeted business service that is running on the JDeveloper 11g integrated WebLogic server.
   - Select the Copy WSDL Into Project option

     Note: If you are using JDeveloper 12c to create a JAX-WS based consumer web-service, a customization file must be used to prevent the WSIMPORT tool from flattening the web service input arguments.
     - Create a customization file like this:

```
<?xml version='1.0' encoding='UTF-8'?>

<jaxws:bindings xmlns:xsd="http://www.w3.org/2001/
XMLSchema"xmlns:jaxb="http://java.sun.com
/xml/ns/jaxb" xmlns:jaxws="http://java.sun.com/xml/ns/jaxws"
xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
wsdlLocation='your WSDL location/URL'>

     <jaxws:enableWrapperStyle>false</jaxws:enableWrapperStyle>

</jaxws:bindings>
```

     - On the Select Web Service Description page, select the Customization Files option, and then select Add File.
     - Browse to the customization file you created and add the customization file to the client creation process.
   - Click Next to open the Specify Default Mapping Options page.
8. On specify Default Mapping Options, do the following:
   - Enter the business service project name package in the Package Name field; for example, oracle.e1.bssv.JRH90134.proxy
   - Enter the business service proxy type in the Root Package for Generated Types field; for example, oracle.e1.bssv.JRH90134.proxy.types
   - Click Next to open the Port Endpoints page.
9. On Port Endpoints, do the following:
   - Verify the system-entered endpoint URL of the targeted JAX-WS based web service.
   - Click Next to open the Asynchronous Methods page.
10. On Asynchronous Methods, select the Don't generate any asynchronous methods option.
11. On the Policy page, click Next to open the Defined Handlers page.

    This page is optional. In this task, a JAX-WS client is being created for the web service running locally, which does not have any security configured. However, if you are creating a JAX_WS client for a web service that is secured with an OWSM policy, select the appropriate policy from the list that appears in the Security tab.
12. On Defined Handlers, accept the system entered information and then click Next to open the Finish page.

13. On Finish, click the Finish button to complete the creation of the JAX-WS business service client proxy.

After the proxy is successfully generated, expand the JAX-WS business service client project in JDeveloper. A folder named Proxy contains the classes required by the proxy to invoke the target web service. This folder also contains a types folder that has all of the classes that are used by proxy to invoke the operations of the target web service.

Next step is to create the *Processor.java class to invoke the target JAX-WS web service. In the client RI JRH90134, refer to the RI_JAXWS_ABWebServiceProcessor.java to see how the Processor class has been coded to invoke the target JAX-WS web service. The RI_JAXWS_ABWebServiceProcessor.java has been coded in such a way so that the same JAX-WS client works against a target RI_AddressBookManager JAX-WS web service running on Jdeveloper 11g's Integrated WLS, Standalone WLS or WAS.

Note the following points regarding the code in the RI_JAXWS_ABWebServiceProcessor.java class:

- Fetch the endpoint URL of the target web service from the softcoding record.
- If the web service is secure, also retrieve user name and password credentials from the softcoding record
- If the target web service is secured with WS-Security and it requires username and password credentials to be passed to the SOAP header from the client, do the following
    a. Create a class that implements the SOAPHandler interface to create the WS-Security Header with the username and password credentials retrieved from the softcoding record.
    b. Create a class that implements the HandlerResolver interface, which ensures that the class that implements the SOAP Handler interface is called.
    c. Create an instance of the SOAPHandler class in the client Processor class, and set the username and password credentials retrieved from the softcoding record in this instance.
    d. Create an instance of the HandlerResolver class in the client Processor class, add the instance of the SOAPHandler class to this instance.
    e. Add the instance of the HandlerResolver class to the instance of the target web service.
    f. Set the endpoint URL, user name, and password credentials retrieved from the softcoding record in the RequestContext object of the target web service.

Note: In the RI-JAXWS_ABWebServiceProcessor.java class, the HeaderHandler class implements the SOAPHandler interface and is used to create the WS-Security Header with the user name and password credentials retrieved from the softcoding record. The HeaderHandlerResolver class implements the HandlerResolver interface and is used to call the HeaderHandler class.

Creating a Deployment Profile for the Business Service Consumer Project

After you create the business service consumer project, you deploy it to the integrated WebLogic server. Before you can deploy the business service consumer project to the Integrated WebLogic server, you create a deployment profile. You use JDeveloper to create a deployment profile for the business service consumer project.

To create a deployment profile:

1. On JDeveloper, right-click on the appropriate business service consumer project and select Project Properties.
2. On Project Properties, do the following:
    - Select the deployment property.
    - Select Use Project Settings as the Deployment option.

- Select the business service consumer project in the Development Profiles area.
- Click Edit.

This action opens the Edit WAR Deployment Profile Properties window.

3. On Edit WAR Deployment Profile Properties, in the tree structure, select File Groups, and then click the New button.
4. On the File Groups page, create a new file group by doing the following:
   - Type WEB-INF in the File Group Name field.
   - Select the Packaging option as the Type of file group.
   - Click OK.

     The WEB-INF file group appears in the tree structure, and the WEB-INF page opens in the right-side pane.
5. On WEB-INF, do the following:
   - Type WEB-INF in the Target Directory in the Archive field.
   - Type WEB-INF in the File Group Name field.
   - Click OK.
6. In the tree structure WEB-IN file Group, select Contributors.
7. In the Contributors pane, do the following:
   - Select the Project Output Directory option.
   - Select the Project Dependencies option
   - In the Order of Contributors field, add the path for the Consumer WEBXML folder; such as, <E1_Install_Path>\system\classes\ConsumerWEBXML.
8. In the tree structure WEB/INF file group, select Filters.
9. In the Filters pane, select the Filters tab and do the following:,
   - Select web.xml.
   - Clear check marks from of the other folders that are under the main folder, Merged Contents of This File Group's Contributors.
   - Click OK.
10. In the tree structure pane, expand the WEB-INF/lib file group and select Contributors.
11. In the Contributors pane, select the SBFProjects in the Libraries Selected for Deployment area, and click OK.
12. In the tree structure pane WEB-INF/lib file group, select Filters.
13. In the Filters pane, select the Files tab and ensure that all of the EnterpriseOne tools foundation jar files are selected, and then click OK.
14. In the tree structure pane, expand the WEB-INF/classes file group and select Contributors.
15. In the Contributors pane, do the following:
    - Select the Project Output Directory option.
    - Select the Project Dependencies option.
    - In the Order of Contributors area, add the SBF path; such as, <E1_Install_Path>\<PathCode>\ini\sbf.
    - Click OK.
16. In the tree structure pane WEB-INF/classes file group, select Filters.
17. In the Filters pane, select the Files tab and do the following:

- Verify that all of the preconfigured EnterpriseOne files (.ini, .properties, .xml, .ora) are selected.
- Expand the oracle folder and verify that only the targeted business service consumer project is selected.
- Click OK.

18. In the tree structure pane, select the Library Dependencies group file.
19. In the Library Dependencies pane, ensure that all of the libraries under the Libraries Selected for Deployment area are selected and click OK
20. In the tree structure pane, expand the Platform group file.
21. In the Platform pane, do the following:
    - Select Weblogic 12.1.x (for JDeveloper 12c) or Weblogic 10.3 (for JDeveloper 11g) from the Default Platform drop-down list.
    - Select IntegratedWebLogicServer from the Target Connection drop-down list.
    - Click OK to open the Deployment window.
22. On the Deployment window, click OK.
23. On JDeveloper, save the changes to the targeted business service consumer project.

Deploying the Business Service Consumer Project to the Integrated WebLogic Server

After you create the deployment profile, deploy the targeted business service consumer project to the integrated WebLogic server.

To deploy the business service consumer project to the Integrated WebLogic server:

1. Start the Integrated WebLogic server.
2. On JDeveloper, right-click on the appropriate business service consumer project, select Deploy, and then select the targeted business service consumer project to deploy to the integrated WebLogic server.
3. Ensure deployment of the business service consumer project is successful by verifying the Deployment Finished message that appears in the Deployment-Log window in the Deployment tab in JDeveloper.

Testing the Business Service Consumer Project

You test your custom business service consumer project in the JDeveloper 11g integrated WebLogic server. The following list identifies additional tasks that you perform before testing your custom business service consumer project.

- The target web service for which the business service consumer project is created is up and running.
- The business service consumer project is successfully deployed and running in the JDeveloper 11g integrated WebLogic server.
- Create a business function that calls your custom business service consumer project.
- Add an OCM record for the Enterprise server to point to the business service server on which your custom business service consumer project is running.
- Add a SYSTEM level service property.
- Create a soft coding record.
- Restart the Business Services server.

After you perform the additional tasks, restart the business services server where the business service consumer project is running. If the business service consumer project is running locally on the JDeveloper 11g integrated WebLogic server, then restart the integrated WebLogic Server.

Log into an EnterpriseOne Web client, and run the test reference implementation AddressBook application, P954001.

Note: You can use reference implementation JRH90I34 (for JAX-WS) or JRH90I33 (for JAX-RPC) as a model for testing your web service proxy. The reference implementation provides set up information for the additional tasks that is specific for the reference implementation. See *"Reference Implementation for Testing the Business Services Server as a JAX-WS Web Service Consumer" in the JD Edwards EnterpriseOne Tools Interoperability Reference Implementations Guide* for more information about Reference Implementation JRH90I34. See *"Downloading and Deploying the JDeveloper 11g Reference Implementation Business Service for WebLogic Server" in the JD Edwards EnterpriseOne Tools Interoperability Reference Implementations Guide* for information about Reference Implementation JRH90I33.

Note: If the target web-service (JAX-WS consumer JRH90I34 is deployed to the integrated WLS of JDeveloper 12c) is secured with SSL, then in the Integrated WLS start script (**startWebLogic.cmd**), add the following argument to the JAVA_OPTION jvm arguments to ignore host name verification.

```
-Dweblogic.security.SSL.ignoreHostnameVerification=true
```

The following is an example:

```
set JAVA_OPTIONS=%JAVA_OPTIONS% -Dweblogic.security.SSL. ignoreHostnameVerification=true
```

typically the start script **startWebLogic.cmd** is at this location: C:\Users\<user_name>\AppData\Roaming \JDeveloper \<system>\DefaultDomain\bin\startWebLogic.cmd

If you have any issues while testing the consumer business service scenario, configure the following logs:

- BSSV log.

  Use the jdelog.properties file in <E1_Install_Path>\ini\sbf folder if the JAX-WS consumer reference implementation is running locally on the JDeveloper 11g integrated WebLogic server.

- Jde and JdeDebug logs in local or for the COK on the Enterprise server, depending where business function B953002 is running.

- IntegratedWeblogicServer log, which is located at:

  C:\Documents and Settings\USER_PROFILE\Application Data\JDeveloper\<system>\DefaultDomain\servers \DefaultServer\logs

Creating a Web Service Proxy Using JDeveloper11g with a JAX-RPC Client

You can create a proxy that consumes a web service that is deployed in the WebLogic server. This section explains how to create the proxy in the development environment.

To select JAX-RPC as the standard for a business service consumer project:

1. In JDeveloper, right-click on the appropriate business service consumer project and select Project Properties.

   The Project Properties window for the selected business service consumer project appears.
2. In the left-hand side tree structure, select Libraries and Classpath.

   The Libraries and Classpath page appears on the right-hand side of the window.
3. In the Classpath Entries area, select JAX-RPC Web Logic Client and JAX-RPC 11 Web Services.
4. Move the JAX-RPC Web Logic Client and JAX-RPC 11 Web Services entries to the top of the page by clicking the Move Up button

   Clicking the Move Up button moves the entry one level at a time; you may need to click the Move Up button multiple times to move the entries to the top of the page.
5. Click OK.
6. On JDeveloper, save the changes that you made to the Libraries and Classpath property of the selected business service consumer project.

To create a web service proxy:

1. From OMW, open JDeveloper 11g.
2. Under the BSSV package (created for the proxy project), create a subfolder named Proxy.

   For example, Oracle uses the following package structure for reference implementations:

   oracle.e1.bssv.JRH90I33.proxy
3. Select the OMW project and then open the project properties.
4. Add the proxy path that corresponds to the package structure in the java source path.

   For example, the following structure might be added to the java source path:

   c:\e812\DV812\Java\source\oracle\e1\bssv\ JRH90I33\proxy
5. Select the BSSV package for which you want to create a proxy, and then click File, and then click New.
6. On New Gallery, select All Technologies from the drop-down menu.
7. Expand the Business Tier category, and then select Web Services.
8. In the Items pane, select Web Services Proxy, and then click OK.

   The Create Web Service Proxy wizard opens.
9. On the Welcome page, click Next.
10. If you receive a prompt to select a web service option, select JAX-RPC, and then click OK.
11. Enter the web service URL address.
12. On the Specify Default Mapping Option page, enter the package name or select the name using the Browse button.

    For example, to use the JRH90I33 reference implementation, select oracle.e1.bssv. JRH90I33.proxy.

13. Click Next.
14. Select the name of the service that you want to call, and then click Next.
15. Verify that the default options are selected on the remaining pages, and then click Finish.
16. Select the newly created proxy folder.

    This folder is the destination folder for the files that are created.

    The proxy files are created.

Rearranging and Renaming Packages

You can compile and build the project you have created.

To rearrange and rename packages:

1. Locate the folder path "proxy\oracle\e1\bssv\JRH90I33\proxy" in the project folder.

   After you copy the files, ensure that you cross check that the package name and the folder structure match.
2. Select the newly created projects in which you place your newly created proxy files in OMW.
3. Select the value object java files that have been generated under the proxy folder.
4. Rename the value object files package name and place it in the appropriate folder.

   You can use the JDeveloper refactor option to change the package name and place the files in the corresponding folder.

   For example, select jpr01000.bssv.e1.oracle.RI_Address and click Refractor. Then, change the package name to:

   oracle.e1.bssv.JRH90I33.proxy.valueobject.oracle.e1.bssv.jpr01000.valueobject.RI_Address
5. Check for all references and change the package structure as appropriate.
6. Compile the project and rebuild the code to ensure no error has occurred.

Note: To test your business service, see *Testing the Business Service Consumer Project*

9 Working with Softcoding

Understanding Softcoding

When you create a business service that enables JD Edwards EnterpriseOne to be a web service consumer, you can use softcoding to dynamically change the endpoint and security information that you provide in the web service proxy. You develop and test your web service against a development environment, which requires an endpoint and possibly security information. The endpoint and user credentials in the production environment will be different from those that you provide in the development environment. In the production environment, the web service proxy needs to know which machine to call for the service, and it needs to know what user credentials, if any, to pass for the external web service call. Softcoding enables you to dynamically plug in the machine name and user credentials at runtime instead of hard-coding them into the business service.

The business service requires a softcoding value at runtime. At a minimum, the softcoding value provides the endpoint for calling an external web service. If user credentials are required by the web service that is being called, the softcoding value includes security information, too. Depending on how many external web services you call, you can have many softcoding values. JD Edwards EnterpriseOne provides softcoding templates and softcoding records to help you manage multiple softcoding values.

Softcoding templates are used by developers during business service development and testing. The purpose of a softcoding template is to define what type of softcoding value is used by each business service. A softcoding template defines the expected structure of the softcoding value and serves as a communication mechanism between a developer and an administrator. Softcoding templates are not used at runtime, but they specify the softcoding value that a business service uses at runtime.

Softcoding records are created by administrators to be used by the system at runtime. A softcoding record contains the actual softcoding value that is to be used by a business service to call an external web service. Typically, softcoding records are created and maintained by an administrator. You can create many softcoding records for a business service. When you create a softcoding record, you provide a softcoding key, which is the name of the business service, a user ID or role information, and the environment.

Understanding Softcoding Applications

JD Edwards EnterpriseOne provides two applications to support softcoding. You use the Web Service Soft Coding Templates program (P953000) to configure web service softcoding templates based on web service security requirements. You use the Web Service Soft Coding Records program (P954000) to configure web service softcoding records based on web service softcoding templates. Both of these applications store XML documents.

Developers create both softcoding templates and softcoding records for testing the business service in the development environment. When testing is finished, the administrator creates softcoding records for the business service to use in the production environment. The softcoding records are based on the softcoding templates that developers created. Typically, softcoding values will be different in the development and production environments.

To improve performance, softcoding records are cached on the business services server at runtime. When you modify a softcoding record, you should also clear the jdbj service cache.

Understanding Encrypted and Dynamic Softcoding Values

Typically, you will want to encrypt your security information. Both softcoding applications (template and records) enable you identify values that are to be encrypted in the XML document. Both applications also enable you to use values that are dynamically replaced in the XML document.

Encrypted Values

You should consider protecting information such as passwords by encrypting the password. Both softcoding applications provide a way to encrypt information within the XML document. In the XML document, the encrypted value has an alias that is surrounded by a specific sequence of characters (_||_).

This example XML document shows a password that you might want to secure:

```
<username-token name="User Name" password="95T763i"
 password-type="PLAINTEXT" add-nonce="false" add-created="false"/>
```

This example code shows how the XML document encrypts the secure value:

```
<username-token name="User Name" password="_||_maskedpassword_||_"
 password-type="PLAINTEXT" add-nonce="false"  add-created="false"/>
```

Both of the softcoding applications provide an area at the bottom of the form for you to enter data to be encrypted. The decryption and substitution occurs at runtime when the getSoftCodingRecord method is called. It is imperative that no logging of the softcoding value takes place between the call to getSoftCodingRecord and _setProperty.

Dynamically Replaced Values

Within a softcoding value, certain values can be used for dynamic replacement. Using these values in the XML means that the values are inserted and returned when getSoftCodingRecord is called. You might use dynamic replacement of values if you have multiple systems that contain the same set of user IDs and login criteria (environment and role) in both systems. You can use these values for dynamic replacement:

- $e1user
- $e1environment
- $e1role
- $ps_token

Creating Softcoding Values

A softcoding value is a segment of an XML document. The simplest softcoding includes only the machine name; no user credentials are required. The softcoding value always includes an endpoint and conditionally includes security information.

This sample code shows softcoding that provides an endpoint only:

```
<port-info>
   <stub-property>
      <name>javax.xml.rpc.service.endpoint.address</name>
      <value>http://www.webservicex.net/WeatherForecast.asmx</value>
   </stub-property>
</port-info>
```

If the web service requires user credentials to be passed, the softcoding contains that information as well.

This sample code shows a softcoding value that provides both endpoint and user credentials:

```
<port-info>
 <stub-property>
   <name>javax.xml.rpc.service.endpoint.address</name>
   <value>http://dens001.mlab.jdedwards.com/Rollout/RI_AddressBookManager</value>
</stub-property>
<wsdl-port namespaceURI="http://oracle.e1.sbf.JPR01000/"
localpart="RI_AddressBookManagerHttpPort"/>
 <runtime enabled=security>
  <security>
   <inbound/>
   <outbound>
    <username-token name="User Name" password="Password" password-type="PLAINTEXT"
add-nonce="false" add-created="false"/>
   </outbound>
  </security>
 </runtime>
 <operations>
   <operation name='addAddressBook'>
  </operation>
   <operation name='addAddressBookAndParent'>
  </operation>
   <operation name='getAddressBook'>
  </operation>
 </operations>
</port-info>
```

You can use JDeveloper to create coding values with many different types of security information.

For more information about Oracle JDeveloper, see the *Oracle Fusion Middleware User's Guide for Oracle JDeveloper.* This document is located under the Fusion Middleware section on the documentation page of the Oracle Technology Network website.

Using Softcoding with Business Service Methods

Softcoding within a business service requires two methods. You use the method getSoftCodingRecord to retrieve the softcoding record. You use the method _setProperty to apply the softcoding record to the web service proxy. Calling _setProperty with a softcoding value overrides hard-coded values that might have been assigned in the web service proxy when it was generated.

This code sample shows how to retrieve a softcoding record and apply the softcoding value to the web service proxy:

```
Element softCodingValue;

// Retrieve the softcoding record. The 'key' that is passed is a string confined
// only by methodology. It becomes part of a multipart key to retrieve a databas
// record.
softCodingValue = SoftCodingRecordAccess.getSoftCodingRecord(context,
                                                             this.SOFTCODING_KEY);

// myPort is the Web Service Proxy
myPort = new RI_AddressBookManagerHttpPortClient();

// apply the softcoding to the proxy if(softCodingValue!= null) {

(Stub)myPort.getPort())._setProperty(oracle.webservices.ClientConstants.CLIENT_
                                                          CONFIG,softCodingValue);
}
else {
   // respond to missing softcoding record. Log, set default values, and/or
return error.
}
```

Managing Softcoding Templates

This section provides an overview of softcoding templates and discusses how to:

- Add a softcoding template for a JDeveloper proxy
- Copy a softcoding template

Understanding Softcoding Templates

You use softcoding templates to create softcoding records. Using a softcoding template is productive because softcoding records have similar values, and reusing templates helps to minimize typing errors when entering record information. Softcoding templates are basically a unique name, a softcoding key that defines a relationship (the value used in the code), and the softcoding value.

A web service proxy has at least one softcoding template and one softcoding record; however, a web service proxy can have many templates and many records. The softcoding key is a grouping mechanism that is used to tie all of the related templates and records together for a given web service proxy. The softcoding key also acts as a communication vehicle for passing information from developers to administrators.

Because the development environment and the production environment are similar but not identical, a developer needs a way to communicate with an administrator. By creating a softcoding template, the developer has a way to communicate both the softcoding key the business service is expecting and the anticipated structure of the XML element.

You use the Web Service Soft Coding Templates Program (P953000) to add new templates, update existing templates, copy existing templates to create new templates, and delete templates.

Forms Used to Manage Softcoding Templates

Form Name	FormID	Navigation	Usage
Work with SoftCoding Templates	W953000A	Type P953000 in the Fast Path.	Locate and review existing templates or delete a template. Deleting a template deletes the mask fields associated with the template.
Add SoftCoding Template	W953000C	On Work with SoftCoding Templates, click Add.	Add a new template.
Update SoftCoding Template	W953000C	On Work with SoftCoding Templates, click Find, select an existing template, and then click Select.	Modify the key and value of an existing template. You cannot change the template name.
Copy SoftCoding Template	W953000C	On Work with SoftCoding Templates, select an existing template, and then click Copy.	Copy an existing template to make a new template. You must change the template name.

Adding a Softcoding Template for a JDeveloper Proxy

Use the Add SoftCoding Template form to add a new softcoding template. There is a standard softcoding template for a JDeveloper proxy that you can use for all the softcoding records. The SoftCoding Template application is P953000.

A generic key is used for all the WebLogic-specific consumer proxies.

Access Add SoftCoding Template form and enter the following information on the form:

Template Name
Enter a name for the template. The template name is a standard name for all proxies.

JD Edwards EnterpriseOne templates are named E1_<BusinessService>, where BusinessService is the name of the business service; for example, J34A0010. Taking into consideration system updates and upgrades, you should not

modify a JD Edwards EnterpriseOne template. Instead, copy the JD Edwards EnterpriseOne template and rename it using a similar naming convention.

You might name a template SC4WLS to denote a proxy named Soft Coding for the WLS server.

Description
Enter text that identifies the purpose of the template.

Softcoding Key
A way to identify related templates and records for a given business service. It is a unique key, such as SC4WLS for the WLS server.

Value
An XML document without the header information; for example, <?xml version=1.0?>.

The following is an example value that you might use as a value for a soft coding template for WLS:

<scwls>

<endpoint>http://host:port/context-root/PortName</endpoint>

<username>username</username>

<password>_||_password_||_</password>

<policy>xyzPolicy.cml</policy>

<trustkey>abcTrustKey.xml</trustkey>

</scwls>

Mask Field
Enter a value that you want to appear in the XML document. This value is an alias for the value that you enter in the Mask Value column. The application automatically places the characters (_||_) around the alias value you enter.

Mask Value
Enter the value for which you have created an alias. This value will appear as the alias value in the XML document.

Copying a Softcoding Template

Use the Copy Softcoding Template form to create a new softcoding template by copying an existing softcoding template. You must provide a new template name.

Managing Softcoding Records

This section provides an overview of softcoding records and discusses how to:

- Add a softcoding record for a JDeveloper proxy.
- Copy a softcoding record.

Understanding Softcoding Records

Softcoding records are used at runtime. They have a multipart key of user/role, environment, and softcoding key. Including environment in the key ensures that production and development entries are not used out of context. Including user/role in the key ensures that softcoding entries can be as specific as necessary. The softcoding key is the part that the business service passes to the getSoftCodingRecord method. The remainder of the information is provided by the context and is used in a hierarchical fashion so that the most specific records are used if available.

At runtime, when the business service passes J34A0010 as the softcoding key, the method looks for the following records in order and stops as soon as one is found, and returns the XML document.

In this example user KB123 has signed in with the role employee to environment PROD:

Sequence	User/Role	Environment	Softcoding Key	Softcoding Value
1	KB123	PROD	J34A0010	XML document segment
2	employee	PROD	J34A0010	XML document segment
3	*PUBLIC	PROD	J34A0010	XML document segment

Forms Used to Manage Softcoding Records

Form Name	FormID	Navigation	Usage
Work with Web Service Soft Coding Records	W954000A	Type **P954000** in the Fast Path field.	Locate and review existing records or delete a record. Deleting a record deletes the mask fields associated with the record.
Add Web Service Soft Coding Record	W954000B	On Work with Web Service Soft Coding Records, click Add.	Add a new record.
Update Web Service Soft Coding Record	W954000B	On Work with Web Service Soft Coding Records, click Find, select an existing record, and then click Select.	Modify an existing record. You cannot change the user/rode, environment, template, or softcoding key.
Copy Web Service Soft Coding Record	W954000B	On Work with Web Service Soft Coding Records, select an existing record, and then click Copy.	Copy an existing record to make a new record. You must change the user/role, environment, and softcoding key.

Add a Softcoding Record for a JDeveloper Proxy

You use the Add Soft Coding Record form to add a new soft coding record. The Soft Coding Records application is P954000.

Access the Soft Coding Record form and enter the appropriate information in the following fields:

User/Role
Enter your JD Edwards EnterpriseOne user ID and role (such as *Public).

Environment Name
Enter the name of the JD Edwards EnterpriseOne environment in which you are working.

Template Name
Enter the template name that you want to use. This is the name that you created using the Softcoding Template application.

SoftCoding Key
Enter a unique key specific to the project.

The value that you enter defines the relationship of this record to other softcoding templates and records. The softcoding key is a way to tie together all related templates and records for a given web service proxy.

SoftCoding Description
Enter a description of the record.

SoftCoding Value
Displays the XML document without header information, such as <?xml version="1.0"?>.

For WLS, change the value to correspond with the project. Use the information below as an example:

<scwls>

<endpoint>https://10.177.115.221:7443/DV812/RI_AddressBookManager?WSDL</endpoint>

<username>JDE</username>

<password>__password__</password>

<policy>Wssp1.2-2007-Https-Usernametoken-Plain.xml</policy>

<trustkey>DemoTrust.jks</trustkey>

</scwls>

The XML document is automatically created when you click the Populate Softcoding Value button that appears on the form when you enter a valid template name.

Mask Field
Enter a value that you want to appear in the XML document. This value is an alias for the value that you enter in the Mask Value column. The application automatically places the characters (_||_) around the alias value you enter.

Mask Value
Enter the value for which you have created an alias. This value appears as the alias value in the XML document.

Copy a Softcoding Record

Use the Copy Web Service Soft Coding Record form to create a new softcoding record by copying an existing softcoding record. When you click Copy, the Add form loads with the Mask Field column cleared. You must provide a new template name.

Applying Softcoding Records

This section provides an overview of using softcoding records at design time and discusses how to configure the web service proxy with a softcoding record.

Understanding Softcoding Records

At design time, you add code to configure the web service proxy with the softcoding record. This involves adding import statements and using the getSoftCodingRecord API to retrieve the softcoding record. This API also replaces dynamic and encoded values if either or both of these types of values are used in the softcoding record.

Configuring the Web Service Proxy with a Softcoding Record

When adding code to configure the web service proxy with the softcoding record, you must add import statements to the XML document that was created by the softcoding record application. This code sample shows import statements for the business service:

```
import oracle.e1.bssvfoundation.util.SoftCodingRecordAccess;
import org.w3c.dom.Element;
import javax.xml.rpc.Stub;
import oracle.e1.bssvfoundation.exception.InvalidSoftCodingRecordException;
```

Next, you use the getSoftCodingRecord API to retrieve the softcoding record. The following sample code shows how to retrieve the softcoding record, where SAMPLE_KEY is the key that is used to retrieve the record:

```
Element softCodingRecord =
     SoftCodingRecordAccess.getSoftCodingRecord(context,"SAMPLE_KEY");
```

The getSoftcodingRecord API throws these two exceptions:

- SoftCodingRecordAccessException
- InvalidSoftcodingRecord Exception.

The API throws a SoftCodingRecordAccessException when a system error occurs during the retrieval of the softcoding record from the JD Edwards EnterpriseOne database. The API throws an InvalidSoftCodingRecordException when the softcoding record is not well-formed, causing a parsing error to occur when the API is converting the softcoding record into a document object model (DOM).

The last step in the process is to configure the web service proxy with the softcoding record. Be sure to check for a null condition before setting the softcoding record. The retrieved softcoding record is null if a softcoding record does not exist for the key. The following sample code shows how to check for a null condition, where the key is SAMPLE_KEY:

```
myPort = new WeatherForecastSoapClient();
if (softCodingRecord!= null)
{

((Stub)myPort.getPort())._setProperty(oracle.webservices.ClientConstants.CLIENT
                                      _CONFIG,softCodingRecord);
}
```

The EnterpriseOne SoftCoding (E1SC) code template is available in JDeveloper for inserting generated code in the business service. To place the template code into the source editor of JDeveloper, type E1SC and press the accelerator (Ctrl+Enter). This will also bring in any associated imports.

10 Working with HTTP Request/Response

Understanding Business Services and HTTP POST

JD Edwards EnterpriseOne enables you to use a business service to communicate with a third-party system using HTTP POST. In this scenario, a business function is invoked by a request from a JD Edwards HTML web client, which in turn calls a business service to make an HTTP POST request. You provide callback information in the posted data for the third-party system to send an asynchronous reply to the request. When the callback reply is received from the third-party system, the published business service, which is included in the callback information, is invoked. The response is returned to the JD Edwards EnterpriseOne HTML web client.

Using Business Services for an HTTP POST Request

This feature uses the correlation data manager and HTTP adapter service. The correlation data manager maintains correlation data, which is the callback information for HTTP communication. You write code for the business service to get correlation data from the Correlation Data Manager. Correlation data can then be passed in your XML payload or in the HTTP header. You use the HTTP Adapter to post data to external sites. HTTP Adapter uses HTTPPostRequest to send data and HTTPPostResponse to receive a response.

HTTP Adapter Methods

This table identifies the classes that you use with the HTTP Post feature:

Method (API)	Description
HTTPAdapterService	Provides utilities to post messages to external HTTP services.
HTTPPOSTRequest	Represents the request format for an HTTP adapter service of a POST operation.
HTTPPOSTResponse	Represents the response format for an HTTP adapter service of a POST operation.
BSSVCorrelationDataManager	Provides access to correlation data. Correlation data provides the callback information for asynchronous HTTP communication.

Correlation Data Management

Correlation data provides the information that the third-party system needs to respond to the JD Edwards EnterpriseOne request. Correlation data consists of this information:

- Callback HTTP listener servlet.
- UserName/environment/role/token for authentication.
- Callback published business service name.
- Callback published business service method.
- DXAPIROUTE information.

Accessing the Correlation Data Manager

The correlation data is compiled into a URL that is provided to the third-party system. The third-party system uses this information to post data to the business service. Access to correlation data is provided through the CorrelationDataManager interface.

This code sample shows how to access the correlation data manager:

```
IBSSVCorrelationDataManager correlationData = context.getBSSVCorrelationData
 Manager();
```

The business service sets the business service name, method, and callback business function in the correlation data. These are the methods that the business service uses:

```
   void setCallbackBSSVName(java.lang.StringbssvName)
   void setCallbackBSSVMethod(java.lang.String bssvMethod)
   void setXAPICallbackBSFN(java.lang.StringxapiCallbackBSFN)
```

This sample code shows how to use these methods:

```
correlationData.setCallbackBSSVName(oracle.e1.bssv.JRH90I33.
RI_HTTPListnerProcessor.java);
  correlationData.setCallbackBSSVMethod(receiveRSSRsponse);
  correlationData.setXAPICallbackBSFN(NotThereYetBSFN);
```

Getting the Callback URL

The business service gets a callback URL from the CorrelationDataManager. This URL is passed to the third-party system so that the third-party system can post data to the business services server. You use this method for the business function to get the callback URL:

```
String getCallbackURL (boolean secureMod)
```

The getCallbackURL method requires a Boolean parameter that indicates whether the callback URL to be created is using secure mode. A secure mode URL is prefixed by *https.* A nonsecure mode URL is prefixed by *http.* This table shows the constants that you use to indicate whether the URL is to be secure:

Constant	Description
USE_NON_SECURE_MODE	Use for secureMode parameter if SSL is not required.

Constant	Description
USE_SECURE_MODE	Use for secureMode parameter if SSL is required.

Placing Correlation Data in the HTTP Header

You can pass correlation data to the third-party system by placing each data item in the HTTP header. This code sample shows the version of the POST method you should use for putting data items in the HTTP header:

```
oracle.e1.bssvfoundation.http.HTTPPOSTResponse postMessage(oracle.e1.
bssvfoundation.http.HTTPPOSTRequest p1, int CorrInHeaderMode)
```

The parameter CorrInHeaderMode specifies whether correlation data needs to be passed in the HTTP header and the format.

This table defines the constants that you can use for the CorrInHeaderMode parameter:

Constant	Description
NO_CORRELATION_IN_HEADER	Tells the method that correlation data is not required in the HTTP header.
CORRELATION_IN_HEADER_WITH_NON_SECURE_CALLBACK	Tells the method to put all correlation data in the HTTP header with a nonsecure (http) callback URL.
CORRELATION_IN_HEADER_WITH_SECURE_CALLBACK	Tells the method to put all correlation data in the HTTP header with a secure (https) callback URL.

This code provides an example of how to use these constants:

```
res = service.postMessage(req, HTTPAdapterService.CORRELATION_IN_
HEADER_WITH_NON_SECURE_CALLBACK);
```

Posting Data to External Sites

You use the HTTPAdapterService to post XML data to external sites. You create the HTTPPostRequest object first. HTTPPostRequest contains the XML data to be posted as well as the URL where data is to be posted. Then, depending on whether you are using HTTP or HTTPS, the appropriate method from HTTPAdapterService is invoked to post the data. This example code shows how to use HTTPAdapterService to post XML data to external sites.

```
//Make HTTP Call
HTTPAdapterService service = context.getHTTPAdapterService();
HTTPPOSTRequest req = new HTTPPOSTRequest(internalVO.getSzPayloadIn());
HTTPPOSTResponse res = null;

req.setUrl(internalVO.getSzURL());

res = service.postMessage(req, HTTPAdapterService.CORRELATION_IN_HEADER_WITH_ NON_#
```

```
SECURE_CALLBACK);
```

Listening for an HTTP Post Response

The business services server uses a listener servlet (HTTPCallbackListenerServlet) to receive incoming messages from third-party systems. Received messages contain callback information, which is used to associate the message with the correct request.

Listener Servlet

The business services server uses the HTTPCallbackListenerServlet to listen for messages from third-party systems. This servlet starts when the business services server is started. The listener servlet listens for incoming XML messages that use HTTP. Third-party web sites must use the callback URL provided by the business services HTTP POST request. The address of the servlet is returned by the getCallbackURL() method. This URL is available in the XML payload or in the HTTP header. The listener servlet is responsible for calling the published business service method that is provided in the callback URL.

Note: The listener servlet does not require any configuration.

HTTPCallbackListenerServlet Process

The listener servlet functions in this way:

- The servlet listens for XML/HTTP messages from external web sites.
- Upon receiving a message, the servlet reads all of the query parameters that were sent as part of the callback URL. If the number of query parameters do not match the number of parameters sent, the servlet throws a ServletException to the third-party web site.
- Next, the query parameters are decoded and the user is authenticated. If authentication fails, the servlet throws a ServletException.
- When authentication succeeds, the servlet sends a response to the external web site indicating that the message was successfully received.

  Note: The servlet does not wait for the response from the published business service call. The servlet returns a standard *message received* response or throws an exception if the message is not received correctly.

- The servlet then checks whether the authenticated user has authority to call the specified published business service method. If the user has authority, the published business service method is called, and the payload is passed in to the value object of the published business service method. The published business service method called from the listener must follow the methodology requirements.

Note:

- *"Understanding Business Services and HTTP POST" in the JD Edwards EnterpriseOne Tools Business Services Development Methodology Guide* .

Sending the Message to the HTML Web Client

The business service can send a notification message to the JD Edwards EnterpriseOne HTML web client. The business service uses this static method in the BSSVSendXAPIMsgToClinet.java class.

```
public static void sendNotifyMsgToClient(IContext context, String szData)
```

This example shows code for sending a notification message:

```
//Make HTTP Call
HTTPAdapterService service = context.getHTTPAdapterService();
HTTPPOSTRequest req = new HTTPPOSTRequest(internalVO.getSzPayloadIn());
HTTPPOSTResponse res = null;

req.setUrl(internalVO.getSzURL());

res = service.postMessage(req, HTTPAdapterService.CORRELATION_IN_HEADER_WITH_
NON_SECURE_CALLBACK);
```

11 Using Service Error Recovery

Understanding Service Error Recovery

You use service error recovery functionality to store the errors that occur after a business function calls a business service. Service error recovery occurs after you call a business service and is used to handle the errors that the calling business function receives from this call. The errors can be from either the jdeCallBusinessService API error codes or from errors in the response (payload) XML.

When one of these types of errors occurs, the system saves data that was generated by the business service in the Services Error Recovery table (F0045). You can then use the Service Error Recovery program (P0045) to review the errors and reprocess the transactions that ended in error. Or you can use the Services Error Recovery batch program (R0045) to reprocess the transactions.

Note: Service error recovery works only for business functions that use the jdeCallBusinessService API. Additionally, service error recovery can be used only if the returning payload contains a tag called <sz-errors>.

When the jdeCallBusinessService API call is successful, but an error occurs during the execution, the data that is saved in the F0045 is the content of the tag <sz-errors>, which is returned in the payload. This string is saved in the BSSVERR field in the F0045. The rest of the data associated with the transactions that end in error are known by the calling business function, which enables the user to reprocess the transaction using the P0045 or R0045 program.

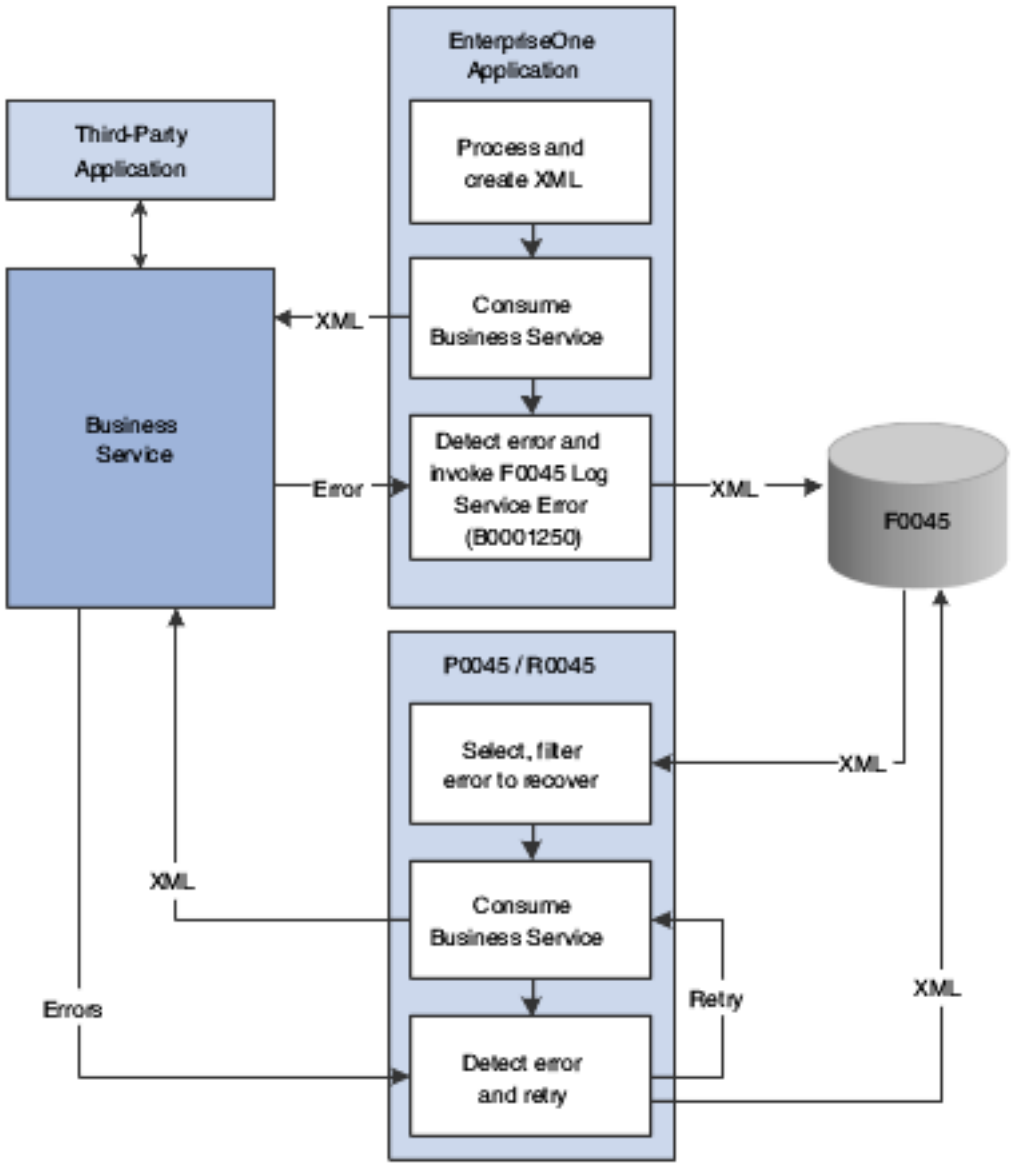
These fields in the F0045 table are used to reprocess the transaction:

- BSSVPCK - BSSV Package
- SBFMDNM - BSSV Method Name
- BSSVXML - Input XML for BSSV
- BSSVTRTY - BSSV Transaction Type

Note: The value object contains a field called szErrors that translates to the <sz-errors> XML tag when returned to the business function in the response XML. This is parsed out of the response XML to show the user the errors that are returned from the business service.

Before you can review service errors, or resend the data that was not successfully delivered by the business service, your business services must be coded to format the error strings that are returned to the business functions. The business function then processes and stores the service errors.

This diagram illustrates the process flow for service error recovery:



The following sections provide an overview of the two parts of the service error recovery flow:

- Recognizing and storing service errors.
- Reviewing errors and resending data.

Recognizing and Storing Service Errors

This section describes how the system recognizes errors and stores them when an error occurs while processing a business service. When you call a business service, it can fail for a number of reasons. The service error recovery flow differs based on the failure reason. The two different flows are based on these types of failure reasons:

- A server connection issue or some other connection issue exists.
- A failure occurs on the third-party end of the process.

In the event of a connection issue:

1. An error status is returned from the business service to the business function.

2. The business function hard-codes the error to `007:FIS`.

   Note: The error 007:FIS is hard-coded for the Order Promising and Requisition Self Service business services, but might be different for other business services.

3. The business function invokes the F0045 Log Service Errors business function (B0001250) to write data to the F0045.

In the event of a third-party error:

1. The service receives errors from the third-party system or system errors.
2. The service translates the error codes from the third-party system to JD Edwards EnterpriseOne error codes (DD alias).
3. The service creates a string in the following format:

   `"n1:alias1|n2:alias2|n3:alias3|...|nx:aliasx"`

   - if nx = 0: error is at header level, system level, or the error is unique
   - nx<>0: transaction detail key where the error occurred
   - nx is DD GENKEY (40 characters)
   - Aliasx: error alias in E1 format
   - Aliasx is DD DTAI (10 characters)

4. The service returns the error string to the calling function by means of the payload in the <sz-errors> tag.
5. The calling function checks the error string.

   If the string is greater than 1,999 characters, the business function truncates the string to 1,999 characters. Some Oracle databases do not support more than two fields greater than 2KB in a table, and the XML BLOB field is already more than 2KB in the F0045 table. If the string is truncated, a truncation flag is set and sent to the error recovery business function.
6. If the string is not null, the business function invokes the F0045 Log Service Errors business function (B0001250) to write data to the F0045 table.

   The error string is stored as is.

The flows differ in the way that they handle the error string. However, both flows use the F0045 Log Service Errors business function (B0001250) to write the error string to the F0045 table. The errors that occur while the system processes the call to jdeCallBusinessService are returned to the user and written to jdedebug.log.

Note: The third party might send back multiple errors for one transaction if those errors occur on different detail records.

Reviewing Errors and Resending Data

After the business function processes the errors, you can review those errors using the Service Error Recovery program (P0045). The Service Error Recovery program:

1. Displays the errors in a readable format.

   The system uses the Parse Service Error String (B0001270) and Service Error Cache (B0001280) business functions when the user reviews the errors.
2. Unparses the error string, retrieves the error descriptions, and displays the details in a grid.

3. If the truncation flag is set, the system displays a message indicating that the error list was truncated and that more details are available in the service logs.
4. Enables you to cancel the transaction or resend the data associated with the transaction.

The system uses the Consume Business Service business function (B0001260) to resend the xml message. Additionally, you can use the Services Error Recovery batch program (R0045) to resend the data.

Note: The error string returned from a service must conform to the format documented here; otherwise, the translation of the error details in the Service Error Recovery program might show unexpected results.

These business functions are used to review and reprocess service error information:

- B0001260 – Consume Business Service

  This business function locks all records in the F0045 table that have the same key (BSSVPCK, SBFMDNM, GENKEY), fetches records for which the unique key identifier matches the identifier in the data structure and consumes the business service for the specified record.

- B0001270 – Parse Service Error String

  This business function parses the BSSVERR error string, which is stored in the F0045 table, and saves records to cache. The function counts the number of error records in the BSSVERR string. It counts the number of pipes "|" and separates the fields when it finds a ":". If more than one record is found, it saves the records in a cache (B0001280). Otherwise, it returns the code and the key.

- B0001280 – Service Error Cache

  This business function is a standard cache-managing business function.

Code Sample: Building the Error String and Mapping It to the Message

When you create your business service, you write code that builds the error string and maps it to the message. This business service code sample illustrates how to build the error string and map it to the message list:

```
//Create error list to concatenate errors for return
       StringBuffer errorList = new StringBuffer();

      //add message to list
                  if (errorList != null) {
                      errorList.append(DELIMITER + msgContext + ":" +
                                       errorsVO.getF34A50_OWERROR());
                  }

        if (!errorList.toString().equals("")) {
           internalVO.setSzErrors(errorList.toString().substring(1));
        }
```

Code Sample: Invoking the F0045 Log Service Error Business Function (B0001250)

The code that invokes the B0001250 business function should be the same or similar for all business functions. This is an example of the C code:

```
/*============================================================
 * Call BSSV API
*============================================================*/
idReturnValue = jdeCallBusinessService(lpDSInternal->lpBhvrCom,
                                       lpDSInternal->lpVoid,
 BSSV_PACKAGE,
 _J("processProcurement"), TRUE,
 szBSSVXMLString,
 &szBSSVPayloadReturn);

if( idReturnValue == CallBSSVNoError)
{
/*  Parse the XML String  */
xrcsStatus = XRCS_parseXMLStringRemoveEncoding(lpDSInternal->hParser,
szBSSVPayloadReturn, &hPayloadDoc);
if(xrcsStatus != XRCS_SUCCESS)
{
jdeTraceSz(NULL, _J("B34A1300 - XRCS_parseXMLStringRemoveEncoding
failed.\n"));
idReturnValue = ER_ERROR;
}
else
{
/* Get Payload Root Element */
xrcsStatus = XRCS_getDocumentElement(hPayloadDoc,&hPayloadRootElm);
if(xrcsStatus != XRCS_SUCCESS)
{
jdeTraceSz(NULL, _J("B34A1300 - XRCS_getDocumentElement failed for
Payload element.\n"));
idReturnValue = ER_ERROR;
}
else
{
* Get Tag Element */
xrcsStatus = XRCS_getElementsByTagName(hPayloadRootElm,
_J("sz-errors"), &hElm,&nElmCount);
   if(xrcsStatus != XRCS_SUCCESS)
{
jdeTraceSz(NULL, _J("B34A1300 - XRCS_getElementsByTagName failed
for szError tag.\n"));
idReturnValue = ER_ERROR;
}
else
{
/* Retrieve tag szErrors from bssvPayloadReturn into szBSSVError string */

if(nElmCount > 0 && hElm != (XRCS_hElement*) NULL)
{
XRCS_getElementText(hElm[0],&szXMLText);
```

```
jdeStrncpyTerminate((JCHAR *)szBSSVError, szXMLText,DIM
(szBSSVError)) ;
idReturnValue = ER_ERROR;
if (jdeStrlen( szXMLText ) > MAX_ERR_LEN)
{
dsD0001250A.cServiceErrorTruncationFlag = _J('1');
}
bCallErrorRecovery = TRUE;
}
else
{
jdeTraceSz(NULL, _J("B34A1300 - Service consumed successfully.\
n"));
}}
}
}/* end else parse error */
} /* ed if no error in BSSV */
else
{
/* Error when connecting to BSSV - Assign "0:007FIS" to BSSVERR */
jdeStrncpyTerminate((JCHAR *)szBSSVError, (const JCHAR *)_J("0:007FIS"),
DIM (szBSSVError)) ;
bCallErrorRecovery = TRUE;
idReturnValue = ER_ERROR;
}

if ( bCallErrorRecovery )
{
/* Call B0001250 */
dsD0001250A.idBSSVDocHandle = (ID)jdeStoreDataPtr(lpDSInternal->hUser,
lpDSInternal->hDoc);
dsD0001250A.cSuppressErrorMessage = _J('1');
jdeStrncpyTerminate(dsD0001250A.szBSSVPackage,
BSSV_PACKAGE,
DIM(dsD0001250A.szBSSVPackage));
jdeStrncpyTerminate(dsD0001250A.szBSSVMethodName,
(const JCHAR *)_J("processProcurement"),
DIM(dsD0001250A.szBSSVMethodName));
jdeStrncpyTerminate((JCHAR *)szPipe, (const JCHAR *)_J("|"),DIM (szPipe));
FormatMathNumeric(szOrderNumber, &lpdsD4302470A->mnOrderNumber) ;
jdeStrcat(szOrderNumber, szPipe);
jdeStrcat(szOrderNumber, lpdsD4302470A->szOrderType);
jdeStrcat(szOrderNumber, szPipe);
jdeStrcat(szOrderNumber, lpdsD4302470A->szOrderCompany);
jdeStrcat(szOrderNumber, szPipe);
jdeStrcat(szOrderNumber, lpdsD4302470A->szOrderSuffix);
jdeStrncpyTerminate((JCHAR *)dsD0001250A.szTransactionKey,
szOrderNumber,DIM (dsD0001250A.szTransactionKey)) ;
dsD0001250A.cActionCode = lpdsD4302470A->cOrderAction;
jdeStrncpyTerminate((JCHAR *)dsD0001250A.szBSSVTransactionType,
(const JCHAR *)_J("AUTO"),DIM (dsD0001250A.szBSSVTransactionType)) ;
jdeStrncpyTerminate((JCHAR *)dsD0001250A.szBSSVError, szBSSVError,DIM
(dsD0001250A.szBSSVError)) ;
jdeStrncpyTerminate((JCHAR *)dsD0001250A.szProgramId,
(const JCHAR *)_J("B34A1300"),DIM (dsD0001250A.szProgramId)) ;
jdeUTime_SetCurrentTime(&dsD0001250A.UniversalDateUpdated);
jdeStrncpyTerminate((JCHAR *)dsD0001250A.szCallingFunctionCode,
lpDS->szCallingBusinessFunction, DIM (dsD0001250A.szCallingFunctionCode));

idReturnValue = jdeCallObject(_J("F0045LogServicesError"),
```

```
NULL, lpDSInternal->lpBhvrCom,
lpDSInternal->lpVoid,&dsD0001250A,
(CALLMAP*)NULL,(int)0,(JCHAR*)NULL,
(JCHAR*)NULL,(int)0);

if (idReturnValue == ER_ERROR)
{
jdeTraceSz(NULL, _J("B34A1300 - Error in call to B0001250:
F0045LogServicesError.\n"));
}

} /* end if bCallErrorRecovery */
}
```

Managing Service Errors

This section provides an overview of service error management and discusses how to:

- Set processing options for the Service Error Recovery programs (P0045 and R0045).
- Review service errors and resend data.
- Run the Services Error Recovery program (R0045).

Understanding Service Error Management

If you are using service error recovery functionality, you can use the Service Error Recovery program (P0045) to determine whether the data being sent by your business service was delivered successfully. If the data was not delivered, the system stores the data, along with an error message detailing the reason that the data was not delivered, in the Services Error Recovery table (F0045). The system recovers the errors for a service in sequential order, and each error record in the F0045 is identified by a unique key. Each F0045 record includes:

- The method name, such as processProcurement.
- The service XML.
- The transaction type, which can be manual or auto.
- Reprocess information, which the system uses to determine how many times the record has been reprocessed.

  The system uses this information so that the service can be consumed until it succeeds, or until a maximum reprocessing count is reached. When the maximum reprocessing count is reached, the system makes the record inactive.

Because different business transaction information and errors are stored, the transaction key is saved as generic text, in a concatenated format. For example, a procurement transaction key is stored as DOCO|DCTO|KCOO|SFXO. The action code for the transaction is also stored in the F0045. Action codes include:

- 1: Add
- 2: Change
- 3: Delete

The description of the error, user-reserved fields, and audit information is also stored in the F0045 table.

You can review these error messages to determine whether issues with your system or process exist, or whether issues with the receiving system exist. You can also attempt to resend the data until the data is delivered successfully.

Resending Data

When you receive error messages, you can resend the data associated with the transaction, or you can delete the transaction. You can resend or delete transactions one at a time, or you can select multiple transactions at once. When you resend the data, the system locks the records that are related to the transactions being processed. You can also specify, using the processing options of the P0045 program, the number of times a record can be reprocessed before the record becomes inactive. If you reach the reprocessing count, and a record becomes inactive before it is sent successfully, you can reactivate the record and attempt to resend it.

Using the Services Error Recovery program (R0045), you can choose to resend all active and inactive records at one time. If you choose to resend inactive records, the system resets those records to active before processing them. Also, if you select a particular record for processing, the system reprocesses all of the records with the same package name in chronological order.

If reprocessing is successful, the record is deleted from the F0045 table, and the P0045 program no longer displays the error message. If the attempt to resend the data was not successful, the reprocess count is updated, and the record remains in the F0045 table.

Record Locking

When you attempt to reprocess records using either the P0045 or R0045 program, the system locks a set of records that are associated with the transaction being processed. The system locks all records with the same service package name (BSSVPCK), service method name (SBFMDNM), and transaction key (GENKEY) as the record being processed. The records are unlocked when processing is complete.

Forms Used to Manage Service Errors

Form Name	FormID	Navigation	Usage
Work With Services Error	W0045A	Daily Operations menu (G34A/ OP/ATO) Services Error Recovery Alternatively, Adv/Tech Operations menu (G43E31), PO Dispatch Error Recovery (P0045)	Review service errors and resend data.
Service Error Recovery Revisions	W0045C	On the Work With Services Error form, select a record and click the View Errors button.	View error details and update error records.

Setting Processing Options for the Service Error Recovery Programs (P0045 and R0045)

You use processing options to define default processing information for a program.

Note: The R0045 and P0045 programs both use this set of processing options. However, the P0045 program uses only option 2 in this set of processing options.

Process

1. Records to Process:

Specify whether the R0045 program processes only active records from the F0045 table, or all active and inactive records. Active records have a value of **A** in the REPSTS field. Inactive records have a value of **I** in the REPSTS field. If you choose to reprocess all records, the system activates all inactive records and changes the reprocess count to 0 (zero) for those records. Values are:

Blank: Process only active records.

1: Process active and inactive records.

2. Reprocess Number:

Specify the maximum number of times that a service can be consumed and fail before the system sets the status to inactive. You must set this processing option to a value greater than zero when the Resend or Delete processing option is blank.

3. Resend or Delete:

Specify whether to resend or delete selected records when running the Service Error Recovery batch program (R0045). Values are:

Blank: Resend selected records.

1: Delete selected records.

Reviewing Service Errors and Resending Data

Access the Work With Services Error form.

BSSV Package (Business Service Package)

Enter the package name of the business service that is associated with the errors that you want to review. For example, to review errors that are generated by the PO Dispatch Processor business service (J43E0030), enter this package name and then click Find:

oracle.e1.bssv.J43E0030.PODispatchProcessor

Rep Num (Reprocess Number)

Review this field to determine how many times the system has attempted to resend the record to the supplier. If the value in this field is greater than the value that is set in the processing options for the allowed number of times to reprocess, the record becomes inactive and the system will not attempt to resend the record. You must manually reactivate the record to reprocess it.

Rep St (Reprocess Status)

Review the value in this field to determine whether the record is active. If the record is active, the system reprocesses the record. If the record is inactive and the R0045 is run, the system ignores the record and does not reprocess it unless you set the processing options of the R0045 program to process inactive records. Values include:

A or Blank: Active

I: Inactive

To resend the record to the external source, select the record, verify that it is active, and click the Resend button.

To review the details of the error associated with a particular record, select the record and then click the View Errors button.

Action Code
Review this field to determine the type of transaction that is in error. Values include:

1: Add a new record.

2: Change an existing record.

3: Delete an existing record.

4: Inquire on an existing record.

Running the Services Error Recovery Program (R0045)

Select Adv/Tech Operations menu (G43E31), PO Dispatch Error Recovery (R0045).

> **Note:** Two menu options are available with this name. One menu option enables you to access the P0045 program, and the other to access the R0045 program. Hover over the menu option with your mouse to view which program is associated with each menu option.

12 Creating Business Services

Understanding Business Services

Published business services are Java classes that manage and run business services. Business services are Java classes that have one or more methods. Business service methods call business functions, database operations, and other business services to provide a specific, described unit of work. Business service methods cannot be exposed as web services for public consumption; instead, business service methods are called by the published business service methods. The published business service methods can be exposed as a web service. JD Edwards provides reference implementations that show how to create both published business services and business services. You can use the reference implementations as models for creating your own services. The reference implementations are not intended for production; they are for reference only.

You might need to add functionality to an existing published business service. You can do this by creating a new business service that performs the task that you require. After you create a new business service, you need to create a new published business service that calls the business service. As an alternative to creating a new business service, you can create a version of the business service and include the new functionality. Creating business service versions is discussed in the methodology guide.

See *"Versioning JD Edwards EnterpriseOne Web Services" in the JD Edwards EnterpriseOne Tools Business Services Development Methodology Guide* .

The concept of value objects is important for understanding JD Edwards EnterpriseOne business services. Value objects are Java classes that manage data. Each published business service method works with two value object classes, an input value object class, and an output value object class. The input and output data are the payload of the web service.

Business service methods use internal value objects. These internal value objects are not published interfaces. A business service method uses one internal value object for both input and output data. The methodology guide provides an overview of value objects for both published business services and business services.

To help you create the Java classes, JDeveloper provides these wizards:

- Published Business Service Class Wizard.
- Business Function Value Object Class Wizard.
- Database Value Object Class Wizard.
- Business Service Class Wizard.

Each of the wizards takes you through a series of steps, prompting you for information or prompting you to select an item. When you complete the series of steps, the wizard generates code that is stored as a Java file in the project that you selected before you started the wizard. The code that the wizard generates is displayed in the JDeveloper edit pane. The generated code has TODO tags to help you complete it. JDeveloper also provides visual aids, such as icons on the left-hand side of the editor and colored indicators on the right-hand side of the editor, to help you complete the code.

The first time that you launch a JDeveloper wizard within a user session, the authentication page appears. If credentials are configured, the authentication page opens with that information. You can overwrite the default credentials.

You can have a published business service class or business service class that contains more than one method. However, the Published Business Service Class Wizard and the Business Service Class Wizard create code for only one method. You add additional methods at the end of the protected method created by the wizard by applying a code

template or by copying and pasting the method created by the wizard. This table identifies the code templates that are available in JDeveloper for you to use with JD Edwards EnterpriseOne business services:

Template	Name	Usage
E1DF	JD Edwards EnterpriseOne Data Formatter	Use when creating a business service to generate code that formats data that comes from the published value object.
E1JD	JD Edwards EnterpriseOne Java Doc for Published Members	Use when creating a published value object class to generate code for creating Javadoc.
E1JDI	JD Edwards EnterpriseOne Java Doc for Internal Members	Use when creating a value object class to generate code for Javadoc.
E1PM	JD Edwards EnterpriseOne Published Business Service Method Call	Use when adding a method to a published business service to generate code for the new public and private methods.
E1SC	JD Edwards EnterpriseOne Configure Web Service Proxy with Soft Coding Record	Use when creating a business service that calls an external web service to generate code for configuring the web service proxy with a softcoding record.
E1SD	JD Edwards EnterpriseOne Add Call to Service Property with Default Value	Use when creating a business service to generate code to call a service property.
E1SM	JD Edwards EnterpriseOne Business Service Method Call	Use when adding a method to a business service class to generate code for the new method.
E1TEST	JD Edwards EnterpriseOne Test Harness Class	Use when creating a test harness main method to test a published business service class.

You access a code template within the Java code. You place your cursor where you want to create the generated code and press Ctrl+Enter. A list of code templates is displayed on a context menu. Select the appropriate template from the list to generate the code.

Note:

- *Business Services Framework*.
- *"Published Business Services" in the JD Edwards EnterpriseOne Tools Business Services Development Methodology Guide* .
- *JD Edwards EnterpriseOne Tools Interoperability Reference Implementations Guide* .

Prerequisites

Before you complete the tasks in this section, verify that:

- JD Edwards EnterpriseOne is installed and running.
- The MTR version of JDeveloper is installed and running.
- JDeveloper is launched from Object Management Workbench (OMW).
- If you use JDeveloper 12c, do not use either JDK 1.6 or JDK 1.7 specific APIs.

Common Elements Used in This Chapter

Back Button
Click this button to return to a previous wizard page. A Back button is on all pages of any JD Edwards EnterpriseOne wizard that you run from JDeveloper.

Cancel Button
Click this button to cancel your entries and to close the wizard. A Cancel button is on all pages of any JD Edwards EnterpriseOne wizard that you run from JDeveloper.

Description
Enter text that describes the JD Edwards EnterpriseOne business function, table, or business view.

Environment
Enter the name of the JD Edwards EnterpriseOne environment in which you are working.

Finish Button
When you click this button, the wizard processes your request, closes the wizard, and returns you to JDeveloper. The Finish button is on all pages of any JD Edwards EnterpriseOne wizard that you run from JDeveloper. This button is unavailable until you have responded to all wizard requests.

Function Name
Enter the name of the business function. On wizard search pages, you can use wildcards when searching for a name, for example, *JDE*.

Help Button
Click this button to access the help feature. A Help button is on all pages of any JD Edwards EnterpriseOne wizard that you run from JDeveloper.

Library
The DLL in JD Edwards EnterpriseOne to which a business function belongs. The library name consists of the DLL name and functional group.

Module/Object Name
The OMW object name of the business function, table, or business view.

Next Button
Click this button to go to the next wizard page. The Next button is on all pages of any JD Edwards EnterpriseOne wizard that you run from JDeveloper.

Object Type
A category that allows you to select either a JD Edwards EnterpriseOne table or business view as a search option. If you completed the Object Name field, you must select the object type that is consistent with the object name. For example,

if you typed FO* in the Object Name field, you must select Table as your object type. If you typed V0* in the Object Name field, you must select business view as your object type.

Password
Enter your JD Edwards EnterpriseOne password.

Reporting System Code
A user-defined code (UDC) that identifies a system or product. This code often is the same as the system code and is used for reporting purposes. On wizard search pages, you can use wildcards when searching for a reporting system code, for example, H*.

Role
Enter your JD Edwards EnterpriseOne role.

System Code
A JD Edwards EnterpriseOne-defined code that identifies a system or product, for example, 42 for Sales Order. On wizard search pages, you can use wildcards when searching for a system code, for example, 4*.

System Code and Product code are used interchangeably.

User Name
The valid name of the individual who is accessing JD Edwards EnterpriseOne.

Adding JDeveloper Projects for Business Services

This section provides an overview of JDeveloper projects for business services and discusses how to add a new project for JD Edwards EnterpriseOne.

Understanding JDeveloper Projects for Business Services

You access JDeveloper from JD Edwards EnterpriseOne OMW. You should have one business service workspace in JDeveloper. This workspace should have been created when JDeveloper was launched from OMW. Each published business service and business service has its own project under the business service workspace, where you can add and modify code for published business services and business services that were created using OMW. The JDeveloper project name matches the business service name in OMW. Naming conventions for JD Edwards business service classes are discussed in the *JD Edwards EnterpriseOne Tools Business Services Development Methodology Guide* . When you create new business services, you develop each published business service and business service within its own project in JDeveloper. A JDeveloper workspace can have many projects.

When you are in OMW, you can create more than one business service object. When you launch JDeveloper, it is launched for a specific business service object, and only the JDeveloper project for that business service is added to the workspace. You can add additional JDeveloper projects for other business services by closing JDeveloper and relaunching it from OMW for other business services or by choosing New EnterpriseOne Project from the workspace context menu. Business services must be added in OMW prior to working with them in JDeveloper.

Sample business services that you can use as a reference for implementing your business services are included in your software delivery. You use OMW and JDeveloper to review the code for these reference implementations, as you would for any other business service.

Note:

- *"Package Naming and Structure" in the JD Edwards EnterpriseOne Tools Business Services Development Methodology Guide* .
- *JD Edwards EnterpriseOne Tools Interoperability Reference Implementations Guide* .

Adding a New Project

To add a new project for a business service:

1. Access the business service workspace in JDeveloper
2. In the navigation pane, right-click the workspace name.
3. From the context menu, click New EnterpriseOne Project.
4. Type the existing OMW project name, and then click Finish.
5. Save the file.

The business service that you previously created in OMW appears in the JDeveloper navigation pane under the business services workspace. The project has a package name—the package prefix is the prefix specified in OMW and the last portion of the package name is the OMW object name. Any Java classes or other artifacts associated with that business service object is included as a part of the JDeveloper project.

Creating Published Business Service Classes

This section provides an overview of published business service classes and discusses how to run the Published Business Service Class Wizard.

Understanding Published Business Service Classes

Published business services are exposed as web services for public consumption. You can create a published business service to meet your specific needs. A published business service is composed of one or more business services, which perform specific tasks. The methodology guide provides naming conventions as well as instructions and guidelines for creating new published business services.

You use the Published Business Service Class Wizard to create the published business service Java class.

Note:

- *"Understanding Published Business Services" in the JD Edwards EnterpriseOne Tools Business Services Development Methodology Guide* .

Running the Published Business Service Class Wizard

The Published Business Service Class Wizard creates a new published business service class by extending the business service framework PublishedBusinessService class. This foundation class, along with other business service framework foundation classes, provides the building blocks for you to create a new published business service class.

The wizard steps you through a series of tasks, prompting you for information for naming the class, the method, the input value object, and the output value object. As a final step, the wizard generates Java code for the published business service class and displays this generated code in the edit pane of JDeveloper.

You can use the prompts, visual aids, and TODO tags to complete the generated code. As previously discussed, JD Edwards EnterpriseOne provides code templates that you can apply to the generated code. If the published business service requires more than one method, you must add code for each additional method.

To run the Published Business Service Class Wizard:

1. In the JDeveloper navigation pane, select the project.
2. Right-click the project.
3. From the context menu, click New.
4. On the new Gallery window, open EnterpriseOne and select Classes, and then select Published Business Service Class.
5. Click OK to launch the wizard.
6. On the Create EnterpriseOne Published Business Service Class page, complete these fields:
    - Name
    - Method Name
    - Input Class
    - Return Class
7. Click OK.
   JDeveloper displays the generated code in the edit pane.
8. Update the generated code.
9. From the JDeveloper File menu, save and exit the code.
   The published business service class is saved in the project that you selected. You can open the class and the code by double-clicking the Java file.

Name

A user-defined designation for the published business service class. This name is usually the description name of the system code with Manager added at the end of the name, for example, AddressBookManager.

Method Name

A user-defined designation for a business service. In the published business service, the method name is the same name as the business service name that is called.

Input Class

A user-defined designation for the input value object class for the published business service. Each business service or business method called by the published business service must have an input class in the published value object.

Return Class

A user-defined designation for the output value object class for the published business service. Each business service or business method called by the published business service must have an output class in the published value object.

Creating Value Object Classes

This section provides an overview of value object classes and discusses how to:

- *Running the Business Function Value Object Class Wizard*
- *Running the Database Operation Value Object Wizard*
- *Running the Media Object Value Object Class Wizard*

Understanding Value Object Classes

Value objects are Java classes that manage data. The input and output parameters of the published business service methods are called value objects. These parameters are the payload of the web service. A method defined in a published business service takes one value object as its input parameter and returns one value object as its output parameter. The input and output parameters of business service operations are called internal value objects. Business service internal value objects are not published interfaces. Business service operations use one internal value object for both input and output.

The business service foundation provides wizards to help you create value object classes that follow methodology rules and guidelines. You use the Business Function Value Object Wizard to create value objects that are based on data structures defined within a business function. You use the Database Operation Value Object Wizard to create value objects that are based on database tables or business views for database operations.

You can use the Media Object Value Object Class Wizard to create value objects that are based on Media Object data structures for Media Object operations.

The methodology guide provides naming conventions as well as rules and guidelines for creating published value object classes and internal value object classes.

Note:

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

Running the Business Function Value Object Class Wizard

You use the Business Function Value Object Wizard to create value objects that are based on data structures defined within a business function.

The Business Function Value Object Wizard guides you through a series of tasks to create value objects based on a business function data structure. You use the wizard to search for and find an existing business function to use as a model. The business function that you select should have all of the input and output parameters that are required for your new value object. You can select all or some of the business function parameters to include in your value object class. At the end of the process, the wizard generates code for the value object.

If you are creating a published business service, each business service method must have an input value object and an output value object. If a published business service calls two or more business services that have the same input or output parameters, the business services can share the appropriate input or output value object.

The wizard provides different ways for you to find an existing business function. On the wizard search page you can use the Find button to return all business functions in JD Edwards EnterpriseOne. You can scroll through all of the business functions and select one. If you have some information about the business function that you want to use, you can enter information in one or more of the search fields to filter the search. You can use wildcards in the search fields. For example, if you know the business function name has a 4 in it, you can use *4* in the Object Name field. If you know the name of the business function, you can use the Advanced Search feature to find the business function.

To run the Business Function Value Object Class Wizard:

1. In the JDeveloper navigation pane, select the project.
2. Right-click the project.
3. From the context menu, click New.
4. On the New Gallery window, open EnterpriseOne and select Classes, and then select Business Function Value Object Class.
5. Click OK to launch the wizard.
6. On the Create EnterpriseOne Business Function Value Object, click Next.
7. Enter credential information if required, and then click Next.
8. On the wizard search page, find a business function by performing one of the following actions:
    - If you do not have any information about the business function, click Find to list all business functions and scroll through the list. Select a business function and then click Next.
    - If you have some information about the business function you want to use, complete one or more of these search fields to filter the list of business functions, and then click Find.

      – Object Name

      – System Code

      – System Code

      – Function Name

      – Reporting System Code

      – Description

      – Library

      You can use wildcards in any of these search fields.

      Select a business function from the search results, and then click Next.
    - If you know which business function you want to use, click Advanced Find.

      Using the drop-down list box, select the appropriate information for each of these fields, and then click Next:

      – Library

      – Module/Object Name

      – Function Name
9. The next page of the wizard opens showing the attributes of the business function that you selected.

   You can sort the names of the parameters by clicking the Name column.
10. Select the parameters that you want to include in your new value object class, and then click Next.

You can select parameters individually by selecting the Include check box in the same row as the parameter that you want. When you select parameters individually, you can display the parameters that you selected by selecting the Display Select Only option. If you want to include all of the parameters in your value object, use the Select All button. Use the Clear All button to clear your selections and start over.

11. On the EnterpriseOne Java Class wizard page, enter the name of the value object that you are creating in the Value Object Name field.

    Value object classes for business services should have the same name as the published business service value object name (input or output) prefaced with the word *Internal.*
12. Select one of these Scope options:
    - Publish - if you are creating a value object class for a published business service.
    - Internal - if you are creating a value object class for an internal business service.
13. Click Finish.

    If the name that you entered for the value object already exists, the wizard sends you a warning message. If the name is a new name, the value object Java file appears in the JDeveloper navigation pane under the project that you selected. Generated code is displayed in the JDeveloper edit pane.
14. Use the visual aids and TODO tags to help you complete the generated code.
15. To create accessors for the members in the value object Java file, right-click anywhere in the generated code.
16. From the context menu, select generate accessors.

    The Generate Accessors dialog window appears.
17. To select all members, click on the top-level check box.
18. Click OK.
19. Save the value object Java file.

Running the Database Operation Value Object Wizard

You use the Database Operation Value Object Wizard to create value objects that are based on database tables or business views. The Database Operation Value Object Wizard guides you through a series of tasks to create value objects based on a database schema. You use the wizard to search for and find an existing table or business view. The table (business view) that you select should have all of the fields that are required for your new value object class. The wizard automatically selects all of the key fields in the table (business view). At the end of the process, the wizard generates code for the value object class and displays the code in the JDeveloper edit pane.

If you are creating a published business service, you run the wizard twice, once to create an input value object class and once to create an output value object class. If the published business service has more than one method, you run the wizard twice for each method. If you are creating a business service, you run the wizard once to create the internal value object, which has both input and output parameters. If the business service has more than one method, you run the wizard multiple times to create an internal value object for each method.

The wizard provides three ways for you to find an existing table (business view). On the wizard search page, you select either the Table or Business View option. You can use the Find button to return all tables or business views in JD Edwards EnterpriseOne, and then scroll through the results and select one. If you have some information about the table or business view that you want to use, you can enter information in one or more of the search fields to filter the search. You can use wildcards in the search fields. For example, if you know the object name has a 4 in it, you can use *4* in the Object Name field. Scroll through the results and select the appropriate table or business view. If you know the name of the table or business view, you can use the Advanced Search feature to find the table or business view.

The methodology guide provides rules and guidelines for creating published value objects and internal value objects for each type of database operation (Select, Insert, Update, and Delete). Naming conventions for the value object classes for both published business service and business service are discussed by database operation type.

To create a database value object:

1. In the JDeveloper navigation pane, select the project.
2. Right-click the project.
3. From the context menu, click New.
4. On the New Gallery window, open EnterpriseOne and select Classes, and then select Database Value Object Class.
5. Click OK to launch the wizard that creates a value object class.
6. On the Create EnterpriseOne Database Value Object Class, click Next.
7. Enter credential information if required, and then click Next.
8. On the wizard search page, select one of these object types:
   - Table
   - Business View
9. Find a table (business view) by performing one of the following actions:
   - If you do not have any information about the table (business view), click Find to list all tables (business views) and scroll through the list. Select a table (business view).
   - If you have some information about the table (business view) that you want to use, complete one or more of these search fields to filter the search, and then click Find:

     – Object Name

     – System Code

     – Description

     – Reporting System Code

     You can use wildcards in any of these search fields.

     Select a table (business view) from the search results, and then click Next.
   - If you know which table (business view) you want to use, click Advanced Find.

     Using the drop-down list box, select the appropriate information for each of these fields, and then click Next:

     – Object Type

     – Object Name
10. The next page of the wizard opens showing the attributes of the table (business view) that you selected.
11. Select the parameters that you want to include in your new value object class, and then click Next.

    You can select parameters individually by selecting the Include check box in the same row as the parameter that you want. When you select parameters individually, you can display only the parameters that you selected by selecting the Display Select Only option. If you want to include all of the parameters in your value object, use the Select All button. Use the Clear All button to clear your selections and start over.
12. On the EnterpriseOne Java Class page, enter the name of the value object that you are creating in the Value Object Name field.
13. Select one of these Scope options:

- Publish – if you are creating a value object class for a published business service.
- Internal – if you are creating a value object class for an internal business service.

14. Click Finish.

    If the name you entered for the value object already exists, the wizard sends you a warning message. If the name is a new name, the value object Java file appears in the JDeveloper editor under the project that you selected.
15. JDeveloper displays the generated code in the edit pane.
16. Use the visual aids and TODO tags to help you complete the generated code.
17. To create accessors for the members in the value object Java file, right-click anywhere in the generated code.
18. From the context menu, select generate accessors.

    The Generate Accessors dialog window appears.
19. To select all members, click the top-level check box.
20. Click OK.
21. Save the value object Java file.

See *"Database Exceptions" in the JD Edwards EnterpriseOne Tools Business Services Development Methodology Guide* .

Running the Media Object Value Object Class Wizard

Use the Media Object Value Object Class Wizard to create value objects that are based on the data structures defined within a Media Object data structure. The Media Object Value Object Class Wizard guides you through a series of tasks to create value objects based on a data structure. You use the wizard to search for and find an existing Media Object data structure to use as a model. The Media Object data structure that you select should have all the parameters that are required for your new value object. You can select all or some of the Media Object data structure parameters to include in your value object class. At the end of the process, the wizard generates code for the value object.

If you are creating a published business service, each business service method must have an input value object and an output value object. If a published business service calls two or more business services that have the same input or output parameters, the business services can share the appropriate input or output value object.

The wizard provides different ways for you to find an existing Media Object data structure. On the wizard search page, you can use the Find button to return all Media Object data structures in JD Edwards EnterpriseOne. You can scroll through all of the Media Object data structures and select one. If you have some information about the Media Object data structure that you want to use, you can enter information in one or more of the search fields to filter the search. You can use wildcards in the search fields. For example, if you know the Media Object data structure name has a 4 in it, you can enter *4* in the Object Name field.

To run the Media Object Value Object Class Wizard:

1. In the JDeveloper navigation pane, select the project.
2. Right-click the project.
3. From the context menu, click New.
4. On the New Gallery window, open EnterpriseOne and select Classes, and then select Media Object Value Object Class.
5. Click OK to launch the wizard.
6. On Create EnterpriseOne Media Object Value Object, click Next.
7. Enter credential information if required, and then click Next.
8. On the wizard search page, find a Media Object data structure by performing one of the following actions:

- If you do not have any information about the business function, click Find and then scroll through the list of available Media Object data structures. Select a Media Object data structure and then click Next.
- If you have some information about the Media Object data structure you want to use, complete one or more of these search fields to filter the list of Media Object data structures, and then click Find.

  - Object Name

  - System Code

  - Reporting System Code

  - Description

  You can use wildcards in any of these search fields.

  Select a Media Object data structure from the search results, and then click Next.

9. The next page of the wizard opens showing the attributes of the Media Object data structure that you selected. You can sort the names of the parameters by clicking the Name column.
10. Select the parameters that you want to include in your new value object class, and then click Next.

    You can select parameters individually by selecting the Include check box in the same row as the parameter that you want. When you select parameters individually, you can display the parameters that you selected by selecting the Display Select Only option. If you want to include all of the parameters in your value object, use the Select All button. Use the Clear All button to clear your selections and start over.
11. On the EnterpriseOne Java Class wizard page, enter the name of the value object that you are creating in the Value Object Name field.
12. Select one of these Scope options:
    - Publish - If you are creating a value object class for a published business service.
    - Internal - If you are creating a value object class for an internal business service.
13. Click Finish.

    If the name that you entered for the value object already exists, the wizard sends you a warning message. If the name is a new name, the value object Java file and an additional valueobject (MOItem_Internal incase of Internal or MOItem_Publish incase of Publish) are created and appear in the JDeveloper navigation pane under the project that you selected. JDeveloper displays generated code for the two value objects in the edit pane.
14. Use the visual aids and TODO tags to help you complete the generated code.
15. To create accessors for the members in the value object Java file, right-click anywhere in the generated code.
16. From the context menu, select generate accessors.

    The Generate Accessors dialog window appears.
17. To select all members, click the top-level check box.
18. Click OK.
19. Save the value object Java file.

Creating Business Service Classes

This section provides an overview of business service classes and discusses how to run the Business Service Class Wizard.

Understanding Business Service Classes

Business service classes are internal only. A business service method provides the business logic for performing a specific task and is exposed to the public by being included in a published business service. A business service method can call one or more business functions or database operations to perform a specific task, and it can call another business service. The methodology guide provides naming conventions as well as rules and guidelines for creating business services.

You use the Business Service Class Wizard to create the business service class.

Running the Business Service Class Wizard

The Business Service Class Wizard creates a new business service class by extending the business service framework BusinessService class. This foundation class, along with other framework foundation classes, provides the building blocks for you to create a new business service class.

The wizard steps you through a series of tasks, prompting you for information for naming the class, the method, and the internal value object. As a final step, the wizard generates code for the business service class and displays this code in the edit pane of JDeveloper.

JDeveloper provides prompts and visual aids help you complete the generated code. The generated code also has TODO tags to help you. If the business service class requires more than one method, you must add code for each additional method. JD Edwards EnterpriseOne provides code templates that you can apply to the generated code. The methodology guide provides naming conventions for the various elements of the business service, and it also provides detail guidance for creating a business service.

To run the Business Service Class Wizard:

1. In the JDeveloper navigation pane, select the project.
2. Right-click the project.
3. From the context menu, click New.
4. On the New Gallery window, open EnterpriseOne and select Classes, and then select Business Service Class.
5. Click OK to launch the wizard.
6. On the Create EnterpriseOne Business Service Class, complete these fields:
    - Name
    - Method Name
    - Input Class
7. Click OK.

   JDeveloper displays the generated code in the edit pane.
8. Update the generated code.
9. From the JDeveloper File menu, save and exit the code.

The business service class is saved in the project you selected. You can open the class and the code by double-clicking the Java file.

Name

Enter a user-defined designation for the business service class. This name is usually a functional description of the method with the word *Processor* added at the end of the name, for example, AddressBookProcessor.

Method Name

Enter a user-defined designation for the operation to be performed. This name is usually the same name as the method in the published business service, for example, addAddressBook.

Input Class
Enter the class name for the internal value object.

Creating Business Function Calls

This section provides an overview of business function calls and discusses how to run the Create Business Function Call Wizard.

Understanding Business Function Calls

A business service method contains the business logic for performing a specific task in JD Edwards EnterpriseOne. You create a business service method that calls a business function. You use the Create Business Function Call Wizard to create a business function call and the accompanying code. The business service method must exist before you can use the Create Business Function Call Wizard.

The methodology guide provides rules and guidelines for creating business services.

Note:

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

Running the Create Business Function Call Wizard

The business service framework provides a Create Business Function Call Wizard that generates Java code for calling a business function. You create a business function call in the code already created by the Create Business Service Wizard. You must be at an appropriate location within the code to create the business call method. A TODO tag should be available to help you find this location. The Create Business Function Call Wizard helps you select a business function to use for creating your new business function call.

To run the Create Business Function Call Wizard:

1. Open the Java file for the business service.
2. In the JDeveloper editor pane, place the cursor at a valid position for inserting the Java logic.
3. From the context menu, select EnterpriseOne, and then select Create Business Function Call.

   If Create Business Function Call is disabled on the context menu, the cursor is not at a valid position in the code.
4. The Business Function Call Wizard is launched.
5. Click Next.

   If the authentication page appears, enter your credentials, and then click Next.

6. On the wizard search page, find an existing business function by performing one of the following actions:
   - If you do not have any information about the business function, click Find to list all business functions and scroll through the list. Select a business function and then click Next.
   - If you have some information about the business function you want to use, complete one or more of these search fields to filter the list of business functions, and then click Find.

     – Object Name

     – System Code

     – Function Name

     – Reporting System Code

     – Description

     – Library

     You can use wildcards in any of these search fields.

     Scroll through the results to find a business function. If the Next Page button is active, click it to see additional results, which are added to the bottom of the list, and continue to scroll.

     If you use the Find button to find a business function, select a business function from the search results and then click Next.
   - If you know which business function you want to use, click Advanced Find.

     Using the drop-down list box, select the appropriate information for each of these fields, and then click Next:

     – Library

     – Module/Object Name

     – Function Name
7. The next page of the wizard opens showing the attributes of the business function that you selected.
8. Select the parameters that you want to include in your new value object class.

   You can select parameters individually by selecting the Include check box in the same row as the parameter that you want. When you select parameters individually, you can display the parameters that you selected by selecting the Display Select Only option. If you want to include all of the parameters in your value object, use the Select All button. Use the Clear All button to clear your selections and start over.
9. Click Finish.

   The Java code is updated with the business function call and accompanying code, which you must complete.

Creating Database Operation Calls

This section provides an overview of database operation calls and discusses how to:

- Run the Create Database Call Wizard.

- Create a Select Database Operation Call.
- Create an Insert Database Operation Call.
- Create an Update Database Operation Call.
- Create a Delete Database Operation Call.

Understanding Database Operation Calls

You can create business services that call select, insert, update, and delete database operations. A select operation retrieves information from JD Edwards EnterpriseOne. Select and query database operations are synonymous. An insert operation adds information to a JD Edwards EnterpriseOne table or business view. An update operation modifies existing information in a JD Edwards EnterpriseOne table or business view. A delete operation removes information from a JD Edwards EnterpriseOne table or business view. You can create a select operation to directly query against the JD Edwards EnterpriseOne tables and business views.

Running the Create Database Call Wizard

The business services framework provides a Create Database Call Wizard that generates Java code for calling a database operation. You create a database call in the code already created by the Create Business Service Wizard. You must be at an appropriate location within the code to create the database call method. A TODO tag should be available to help you find this location. The Create Database Call Wizard helps you select a table or business view to use for your new database operation call. After you launch the wizard, it displays a page that has the following database operations:

- Select
- Insert
- Update
- Delete

After you select the type of database operation you want to use, the wizard helps you search for a table or business view to use in your database operation call. The wizard then guides you through a series of steps prompting you for information.

After you specify the appropriate information, the wizard creates a conceptual SQL statement that you can preview. If the SQL statement does not show what you need, you can return and change your specifications. When you are satisfied with your selections, the Database Call Wizard generates the call and accompanying code, which is unique for the database operation that you selected.

The Create Database Call Wizard allows the data types and operations identified in this table:

Data Type	Data Type Shown in Wizard	Allowed Operators
EVDT_CHAR	Character	=, >, >=, <, <=, <>
EVDT_JDEDATE	Date	=, >, >=, <, <=, <>
EVD_INT	Integer	=, >, >=, <, <=, <>
EVDT_LONGVARCHAR	Character (BLOB)	Not allowed in WHERE clause

Data Type	Data Type Shown in Wizard	Allowed Operators
EVDT_LONGVARBINARY	Binary (BLOB)	Not allowed in WHERE clause
EVDT_STRING	String	=, >, >=, <, <=, <>, LIKE If size is greater than 255, Oracle database treats as a BLOB, which is not allowed in WHERE clause
EVDT_VARSTRING	Variable String	Not allowed in WHERE clause
EVDT_JDEUTIME	JDE Utime	=, >, >=, <, <=, <>
EVDT_MATH_NUMERIC	Numeric	=, >, >=, <, <=, <>
EVDT_ID	Identifier (ID)	=, >, >=, <, <=, <>

To run the Database Call Wizard:

1. Open the Java file for the business service.
2. In the JDeveloper editor pane, place the cursor at a valid position for inserting the Java method.
3. From the context menu, select EnterpriseOne, and then select Create Database Call.

   If Create Database Call is disabled on the context menu, the cursor is not at a valid position in the code.
4. Click Next.
5. On the Create EnterpriseOne Database Call introduction window, click Next.

   If the authentication page appears, enter your credentials, and then click Next.
6. On the Select a Database Operation page, select a database operation and then click Next.
7. On the wizard search page, select one of these object types:
   - Table
   - Business View
8. Find a table (business view) by performing one of the following tasks:
   - If you do not have any information about the table (business view), click Find to list all tables (business views) and scroll through the list. Select a table (business view).
   - If you have some information about the table (business view) you want to use, complete one or more of these search fields to filter the search, and then click Find:

     – Object Name

     – System Code

     – Description

     – Reporting System Code

     You can use wild cards in any of these search fields.

Select a table (business view) from the search results, and then click Next.

- If you know which table (business view) you want to use, click Advanced Find.

  Using the drop-down list box, select the appropriate information for each of these fields, and then click Next:

  – Object Type

  – Object Name

9. The next page of the wizard opens showing the operation that you selected along with the attributes of the table (business view) that you selected.
10. Depending on the database call operation you are creating, go to one of these tasks in this guide:

    - Create a Select database operation.
    - Create an Insert database operation.
    - Create an Update database operation.
    - Create a Delete database operation.

Creating a Select Database Operation Call

If you selected the Select Database Operation, the wizard opens the Select Operation main page and displays the attributes of the table (business view) that you selected for the Select operation. The Select operation main page has three tabs: Select Columns, Where Clause, and Order by Columns.

Select Columns Tab

The Select Columns tab shows all of the columns that are available in the table or business view that you selected. Each column from the table (business view) is displayed as a row, and includes the column name, description of the column, and column type, and indicates whether the column is a primary key. You can sort the rows by clicking in any of the column headers. For example, when you click the column header Column Name, you can sort the rows in ascending, descending, or default order by column name.

From the table (business view), you select the columns that you want to use in your Select database operation. You can select specific columns by selecting the check box in the appropriate row, or you can select all of the columns by clicking the Select All button. If you did not select all columns, you can select the Display Selected Only option to show those columns that you selected. You can clear all of your selections by clicking the Clear All button.

The Fetch All Records option and Select Distinct option are used by the system at runtime. The Fetch All Records option works with tables and business views and fetches all records that meet the search criteria. The Select Distinct option works with both tables and business views, but is more commonly used with business views. When many records meet the search criteria, you can use the Select Distinct option to select only one record for the search criteria instead of selecting all of the records. If you do not select this option, then the system returns all occurrences of these records, including all detail lines.

Where Clause Tab

You use the where clause to filter the data for your Select database operation. You can set conditions for one or more of the column names that you have selected. To add a where clause, click the Add Where Clause button. After you add a column, you can click the column name and select a new name from the drop-down list box. To remove a where clause from your operation, select the attribute that you want to remove and then click the Remove Where Clause button. You

change the condition (and/or) by clicking the condition in the row that you want and selecting a condition from the drop-down list box. You change the operator by clicking the operator in the row that you want to change and selecting an operation from the drop-down list box.

The Exclude If Value Is Null option is used by the system for code generation. If you select this option, conditional logic is added to the generated code to not include fields with a null value. If a field has a null value, the field is not included in the where clause. The system builds the where clause using only those value object fields that are not null.

Order by Columns Tab

Use the Order by Columns tab to specify how you want the system to display the results. You specify sort criteria for the columns that you have selected and the order of the records for each column. Click the Add ordering button to add a column name, then change the column name by clicking the column name in the row that you want to change and selecting a new name from the drop-down list box. Use the Move Up and Move Down buttons to change the order of the column names that you enter. You can specify the order of the records within a column to be ascending or descending by selecting the row that has the column name and then clicking the ASC or DESC name under the Ordering column and selecting a new order from the drop-down list box.

After you complete the specifications for your Select call, click Next to see a preview page. The preview page shows a conceptual SQL statement based on the specifications you have entered. You can use the Back button to return to a previous page and modify your specifications. When you are satisfied with the specifications you have entered, click Finish. The wizard creates a Select database operation call where your cursor is located in the business service code, and it produces accompanying code. You must modify the Select call to include the appropriate value objects and to complete the generated code.

Creating an Insert Database Operation Call

If you selected the Insert database operation, the wizard opens the Insert Operation main page and displays information from the table or business view that you selected. The Insert operation main page has only one tab, Insert Columns.

The Insert Columns tab shows all of the columns that are available in the table (business view) that you selected. Each column from the table (business view) is displayed as a row and includes the column name, description of the column, column type, and column length, and indicates whether the column is a primary key. You can sort the rows by clicking in a column header. For example, when you click the Column Name header, you can sort the rows in ascending, descending, or default order by column name. You can sort by any one of the column headers.

From the table or business view, you select the columns that you want to use in your Insert database operation. You can select specific columns by selecting the check box in the appropriate row, or you can select all of the columns by clicking the Select All button. If you did not select all columns, you can select the Display Selected Only option to show only those columns that you selected for your Insert operation. You can clear all of your selections by clicking the Clear all button.

After you select the columns you want to use in your Insert database operation, click Next to see a preview page. The preview page shows a conceptual SQL statement. You can use the Back button to return to a previous page and select or clear columns from the table or business view. When you are satisfied with your selections, click Finish. The wizard creates an Insert database operation call where your cursor is located, and it produces accompanying code. You must modify the Insert call to include the appropriate value objects and you should review the generated code and update as necessary.

Creating an Update Database Operation Call

If you selected the Update database operation, the wizard opens the Update operation main page and displays information from the table or business view that you selected. The Update operation main page has two tabs, Update Columns and Where Clause.

Update Columns Tab

The Select Columns tab shows all of the columns that are available in the table (business view) that you selected. Each column from the table or business view is displayed as a row, and includes the column name, description of the column, and column type, and indicates whether the column is a primary key. You can sort the rows by clicking in a column header. For example, when you click the Column Name header, you can sort the rows in ascending, descending, or default order by column name.

Where Clause Tab

You use the where clause to set conditions for the attributes in your database operation. To add a where clause, click the Add Where Clause button. After you add a column, you can click the column name and select a new name from the drop-down list box. To remove a where clause from your operation, select the attribute that you want to remove and then click the Remove Where Clause button. You change the condition (and/or) by clicking the condition in the row that you want and selecting a condition from the drop-down list box. You change the operator by clicking the operator in the row that you want to change and selecting an operation from the drop-down list box.

After you complete the specifications for your Update call, click Next to see a preview page. The preview page shows a conceptual SQL statement based on the specifications you have entered. You can use the Back button to return to a previous page and modify your specifications. When you are satisfied with the specifications you have entered, click Finish. The wizard creates an Update database operation call where your cursor is located, and it produces accompanying code. You must modify the Update call to include the appropriate value objects and you must complete the generated code.

Creating a Delete Database Operation Call

If you selected the Delete database operation, the wizard opens the Delete operation main page, which contains no data, but has a Where Clause tab. You add a where clause by clicking the Add Where Clause button. After you add a column, you can click the column name in the row where you want to change the name and select a new name from the drop-down list box. To remove a where clause from your operation, select the attribute that you want to remove and then click the Remove Where Clause button. You change the condition (and/or) by clicking the condition in the row you want and selecting a condition from the drop-down list box. You change the operator by clicking the operator in the row you want to change and selecting an operation from the drop-down list box.

After you specify the information that you want to include in your Delete operation, click Next to see a preview page. The preview page shows a conceptual SQL statement. If you click Next without specifying a where clause, the wizard sends you a warning message. You can use the Back button to return to the Delete operation main page and modify your specifications. When you are satisfied with the specifications you have entered, click Finish. The wizard creates a Delete database operation call where your cursor is located, and it produces accompanying code. You must modify the Delete call to include the appropriate value objects and you should review the generated code and make necessary changes.

Creating Media Object Operation Calls

This section provides an overview of Media Object operation calls and discusses how to run the Create Media Object Call Wizard.

Understanding Media Object Operation Calls

You can create business services that call select, insert, and delete Media Object operations. A select operation retrieves media objects from JD Edwards EnterpriseOne. An insert operation adds media objects and the information contained in the media objects to JD Edwards EnterpriseOne. A delete operation removes information and media objects from JD Edwards EnterpriseOne. For rules and guidelines for creating business services, see the *JD Edwards EnterpriseOne Tools Business Services Development Methodology Guide* .

Running the Create Media Object Call Wizard

The business service framework provides a Create Media Object Call Wizard that generates Java code for calling a Media Object operation. You create a Media Object call in the code already created by the Create Business Service Wizard. You must be at an appropriate location within the code to create the business call method. A TODO tag should be available to help you find this location. The Create Media Object Call Wizard helps you select an Operation to use for creating your new Media Object call.

To run the Create Media Object Call Wizard:

1. Open the Java file for the business service.
2. In the JDeveloper editor pane, place the cursor at a valid position for inserting the Java logic.
3. From the context menu, select EnterpriseOne, and then select Create Media Object Call. If Create Media Object Call is disabled on the context menu, the cursor is not at a valid position in the code.

   The Create Media Object Call Wizard is launched.
4. Click Next. If the authentication page appears, enter your credentials, and then click Next.
5. On the Operation Selection page, select one of the following operations:
   - Select
   - Insert
   - Delete
6. Click Finish.
7. The Java code is updated with the Media Object call and accompanying code, which is unique for the Media Object operation that you selected.
8. Manually change the ValueObject type "InternalVO" in the generated Method signature to the value object type you created using the Media Object Value Object Class Wizard.

13 Appendix A - Configuring JDeveloper to Support UTF-8

Understanding UTF-8

If you use non-English characters or data in your business services, you must configure JDeveloper to support UTF-8. When you set up preferences and default project properties to support UTF-8 encoding, all existing projects and any new projects that you add show the preferences and the default project property as UTF-8 encoding. You must individually set up each project where you have non-English characters or data to show UTF-8 as the encoding value.

You are not required to set up UTF-8 encoding when you test web services.

Configuring Preferences

To configure preferences to support UTF-8:

1. On JDeveloper, select the Applications folder or any project folder and then select Preferences from the Tools menu.
2. On Preferences, select Environment.
3. In the Environment pane, select **UTF-8** from the Encoding drop-down list box.
4. Click OK.

   All projects in the workspace show UTF-8 as the value for the Encoding field when you select Preferences from the Tools menu.

Configuring Default Project Properties

To configure default project properties to support UTF-8:

1. On JDeveloper, select the Applications folder, the workspace folder, or any project folder, and then select Project Properties from the Tools menu.
2. In the navigation pane on Project Properties, select Compiler.
3. In the Compiler pane, select **UTF-8** from the Character Encoding drop-down list box.
4. Click OK.

   All projects in the workspace show UTF-8 as the value for the Character Encoding field when you select Default Project Properties from the Tools menu.

Configuring a Project

To configure a project to support UTF-8:

1. On JDeveloper, right-click the project folder that will include non-English data, and then select Project Properties from the context menu.
2. In the navigation pane on Project Properties, select Compiler.
3. In the Compiler pane, select **UTF-8** from the Character Encoding drop-down list box.
4. Click OK.

   Only the selected project shows **UTF-8** as the value for the Character Encoding field.
5. Restart JDeveloper.

You can insert non-English data and run your project.

14 Appendix B - Testing a Business Service That Consumes an External Web Service

Creating a Test Business Service

To create a business service that tests the business service that calls a web service and passes an XML file:

1. Create a test business service object in OMW.
2. Move the generated sample XML file to the test business service folder.

   See *Generating a Sample XML Document*
3. Add a call to the TestServiceBusinessFunction.callBSSVWithXMLFile method.

You can use this example code to create a business service to test calling a business service that passes an XML file.

```
Package oracle.e1.sbf.JTR"H90I10;

Import oracle.e1.sbffoundation.base.TestServiceBusiness Function;
Import oracle.e1.sbffoundation.connection.Iconnection;

Public class RI_AsyncSendEmailProcessor Test {
     Public RI_AsyncSendEmailProcessorTest() {
     }

     public static void main(String[] args) {
        Try {
          //call required prior to executing test from application (main())
          TestServiceBusinessFunction.startTest();
          String file = "C:\\B9\STAGINGA\\java\\source\\oracle\\e1\\sbf\\
JTRH90I10\\SendEmailVO1.xml";

TestServiceBusinessFunction.callSBFWithXMLFile("oracle.e1.sbf.JRH90I20.
RI_AsyncSendEmail Processor",
                                               "sendEmail";
                                               File,
                                               IConnection.AUTO);
         } finally {
           //call required at the end of testing from application (main())
           TestServiceBusinessFunction.finishTest();
         }
     }
}
```

Using the Development Business Services Server

You can do end-to-end testing of your outgoing web service calls on a development business services server. You deploy the business service that calls a web service to a WebSphere Express instance. You can then use the web development client to run the application that calls the business function, which in turn calls the business service. You

can perform the entire test from your web development client. The development business services server uses the WebSphere Express instance that is being used to run HTML applications on the web development client.

Before you deploy the development business services server, you must prepare the configuration files. This topic also discusses how to start and stop the development business services server.

Prerequisites

Before you complete the tasks in this section, verify that:

- The web development client is installed and operational.

  The development business services server and web development client use the same application (WebSphere Express).
- You have a business function that calls a business service method, and you have an application that calls the business function.
- OCM is configured to send business service messages to a local business services server.

Preparing Configuration Files

The development business services server packages the business services server development configuration files, which are in <Path_Code>/ini/sbf), in an EAR file to be deployed.

You edit the JDELOG.PROPERTIES file to get the log files in the required path. The default JDELOG.PROPERTIES file that is installed with the development business services server provides a relative path to the log folder. The JDELOG.PROPERTIES file is in the <Path_Code>/ini/sbf folder. The relative path in the file is evaluated from this folder to get to the log folder location. The relative path changes when you are working with the development business services server. Because the JDELOG.PROPERTIES file is packaged in an EAR file and the EAR file is deployed in a WebSphere instance, the relative path to the log folder may not be valid. Oracle recommends that you use the complete path in JDELOG.PROPERTIES when working with the development business services server.

Deploying a Development Business Services Server

You deploy a development business services server from the JDeveloper Applications Navigator. Right-click the business service workspace, and then select Deploy Development BSSV Server from the context menu.

A dialog box for providing HTTP proxy configurations appears.

The HTTP proxy server parameters that were previously set in the configuration file are displayed. You can change any of these parameters. If you change the HTTP proxy parameter values while the business services server is running, the server might send a prompt indicating that the server will be restarted. Any existing application sessions on the web development client will be terminated. You should save all work before continuing with the restart.

Deployment time varies based on the various parameters. If you are using WebSphere and deploying for the first time, profile creation could take up to 10 minutes. The deploy status is visible in the JDeveloper status window with the title Apache Ant - Log. You can continue with other work while deployment is in progress. You must not stop the JD Edwards EnterpriseOne Solution Explorer (which also stops the WebSphere Express application) while deployment is in progress.

If the development business services server is already installed on the business services server, the previous version is automatically undeployed before the system continues the deployment process. When the deployment is finished, the

Apache Ant - Log window displays a message indicating that the build and deploy was successful. The development business services server is now started and ready to accept requests.

Start or Stop a Development Business Services Server on WebSphere Express

On WebSphere Express, the business services server runs in a server profile called DEVBSSVSvr. This is separate from the application server instance for HTML applications, which runs in default profile. Therefore, starting and stopping JD Edwards Solution Explorer has no effect on the business services server. The easiest way to restart the business services server is to redeploy using the Deploy menu item.

Use this code to stop the development business services server:

```
<WebSphere Install Location>/profiles/DevBSSVSvr/bin/stopserver server1
```

Use this code to start the development business services server:

```
<WebSphere Install Location>/profiles/DevBSSVSvr/bin/startserver server1
```

15 Appendix C - Business Services Framework Javadoc

Understanding Business Services Framework Javadoc

Javadoc is a tool that parses the declarations and documentation comments in a set of Java source files and produces a corresponding set of HTML pages. The Javadoc for the business services framework (foundation packages) is generated during the package build process, and it is included in your software delivery. You can use the following path on your deployment server to access the Javadoc files:

```
<PACKAGE_NAME>\java\sbf\javadoc
```

You also view the machine-generated documentation in HTML format by extracting the contents of the SBFJavadoc.jar file. This is an example of the path you would follow to access the jar file:

```
C:\B9\System\Calsses\SBFJavadoc.jar
```

Note:

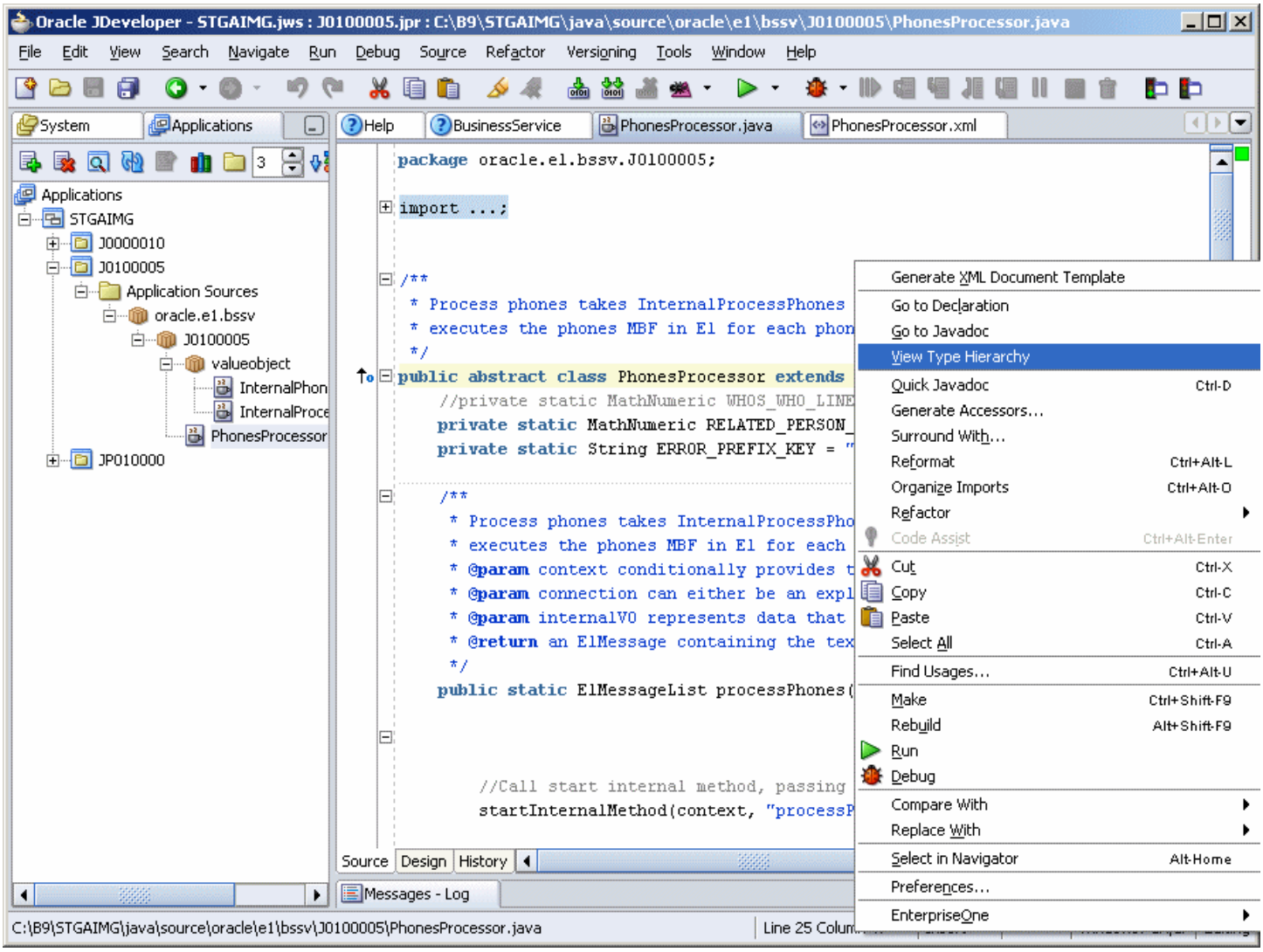
- *Business Services Framework*.

Reviewing Business Services Framework Javadoc from JDeveloper

You can review business services foundation classes from within JDeveloper for all business services projects.

To review a foundation class within JDeveloper:

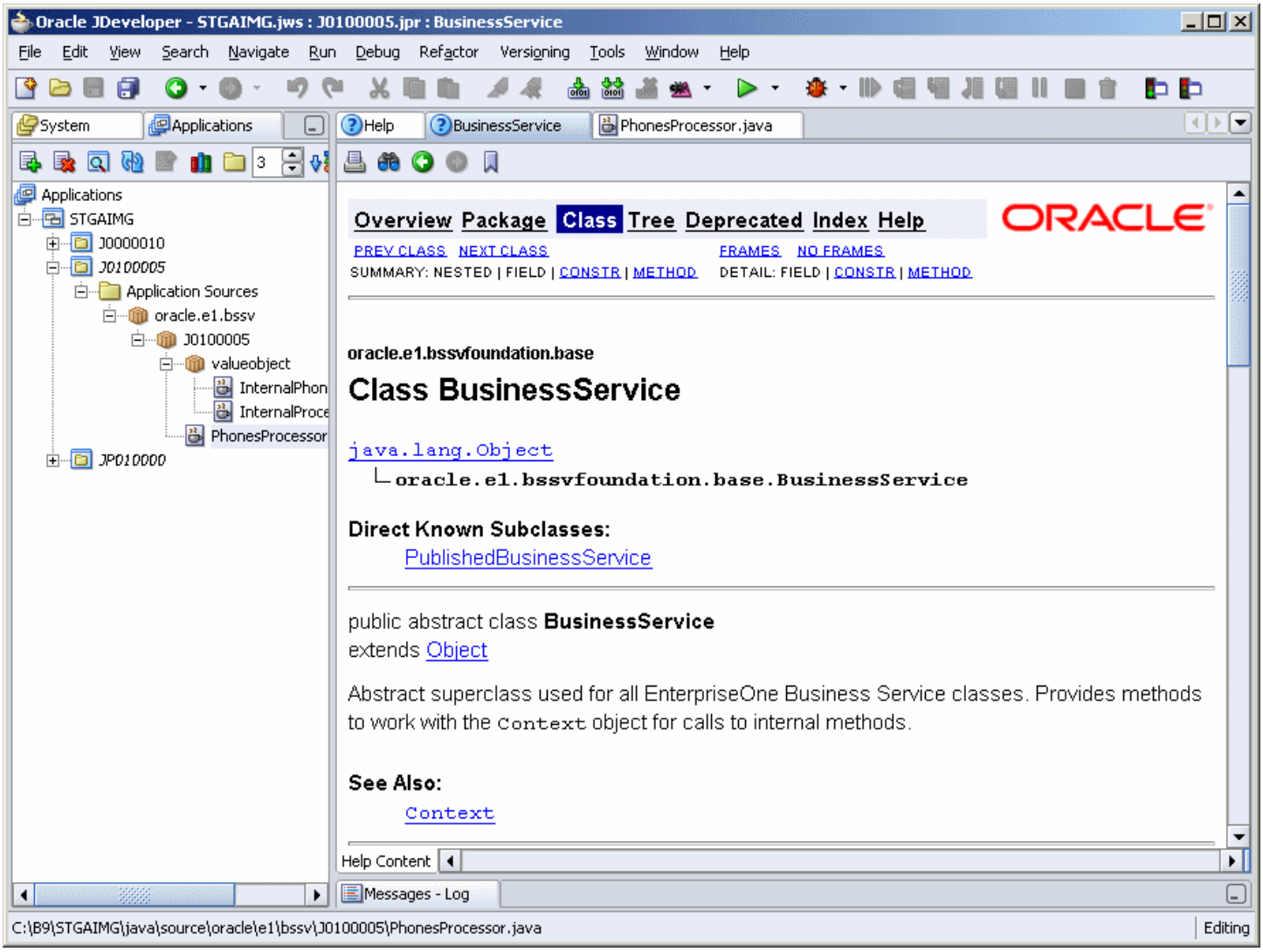
1. In JDeveloper, select a business service.


Oracle JDeveloper - STGAIMG.jws : J0100005.jpr : C:\B9\STGAIMG\java\source\oracle\e1\bssv\J0100005\PhonesProcessor.java
File Edit View Search Navigate Run Debug Source Refactor Versioning Tools Window Help
System
Applications
Help
BusinessService
PhonesProcessor.java
PhonesProcessor.xml
Applications
STGAIMG
J0000010
J0100005
Application Sources
oracle.e1.bssv
J0100005
valueobject
InternalPhon
InternalProce
PhonesProcessor
JP010000
package oracle.e1.bssv.J0100005;
import ...;
/**
* Process phones takes InternalProcessPhones
* executes the phones MBF in E1 for each phon
*/
public abstract class PhonesProcessor extends
//private static MathNumeric WHOS_WHO_LINE
private static MathNumeric RELATED_PERSON_
private static String ERROR_PREFIX_KEY = "
/**
* Process phones takes InternalProcessPho
* executes the phones MBF in E1 for each
* @param context conditionally provides t
* @param connection can either be an expl
* @param internalVO represents data that
* @return an ElMessage containing the tex
*/
public static ElMessageList processPhones(
//Call start internal method, passing
startInternalMethod(context, "processP
Source Design History
Messages - Log
C:\B9\STGAIMG\java\source\oracle\e1\bssv\J0100005\PhonesProcessor.java
Line 25 Colum
Generate XML Document Template
Go to Declaration
Go to Javadoc
View Type Hierarchy
Quick Javadoc Ctrl-D
Generate Accessors...
Surround With...
Reformat Ctrl+Alt-L
Organize Imports Ctrl+Alt-O
Refactor
Code Assist Ctrl+Alt-Enter
Cut Ctrl-X
Copy Ctrl-C
Paste Ctrl-V
Select All Ctrl-A
Find Usages... Ctrl+Alt-U
Make Ctrl+Shift-F9
Rebuild Alt+Shift-F9
Run
Debug
Compare With
Replace With
Select in Navigator Alt-Home
Preferences...
EnterpriseOne

2. In the JDeveloper editor pane, right-click on a foundation business service class name, and then click Go to Javadoc from the context menu, as shown in the preceding diagram.

   Other examples of foundation business service class names are PublishedBusinessService (which must be extended by every published business service) and BusinessServiceException.

The following example shows the foundation Javadoc.


Oracle JDeveloper - STGAIMG.jws : J0100005.jpr : BusinessService
File Edit View Search Navigate Run Debug Refactor Versioning Tools Window Help
System
Applications
Help
BusinessService
PhonesProcessor.java
Applications
STGAIMG
J0000010
J0100005
Application Sources
oracle.e1.bssv
J0100005
valueobject
InternalPhon
InternalProce
PhonesProcessor
JP010000
Overview Package Class Tree Deprecated Index Help
PREV CLASS NEXT CLASS
FRAMES NO FRAMES
SUMMARY: NESTED | FIELD | CONSTR | METHOD
DETAIL: FIELD | CONSTR | METHOD
oracle.e1.bssvfoundation.base
Class BusinessService
java.lang.Object
oracle.e1.bssvfoundation.base.BusinessService
Direct Known Subclasses:
PublishedBusinessService
public abstract class BusinessService
extends Object
Abstract superclass used for all EnterpriseOne Business Service classes. Provides methods to work with the Context object for calls to internal methods.
See Also:
Context
Help Content
Messages - Log
C:\B9\STGAIMG\java\source\oracle\e1\bssv\J0100005\PhonesProcessor.java
Editing

16 Glossary

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.

business service

EnterpriseOne business logic written in Java. A business service is a collection of one or more artifacts. Unless specified otherwise, a business service implies both a published business service and business service.

business service artifacts

Source files, descriptors, and so on that are managed for business service development and are needed for the business service build process.

business service class method

A method that accesses resources provided by the business service framework.

business service configuration files

Configuration files include, but are not limited to, interop.ini, JDBj.ini, and jdelog.properties.

business service cross reference

A key and value data pair used during orchestration.

business service cross-reference utilities

Utility services installed in a BPEL/ESB environment that are used to access JD Edwards EnterpriseOne orchestration cross-reference data.

business service development environment

A framework needed by an integration developer to develop and manage business services.

business services development tool

Otherwise known as JDeveloper.

business service EnterpriseOne object

A collection of artifacts managed by EnterpriseOne LCM tools. Named and represented within EnterpriseOne LCM similarly to other EnterpriseOne objects like tables, views, forms, and so on.

business service framework

Parts of the business service foundation that are specifically for supporting business service development.

business service payload

An object that is passed between an enterprise server and a business services server. The business service payload contains the input to the business service when passed to the business services server. The business service payload contains the results from the business service when passed to the Enterprise Server. In the case of notifications, the return business service payload contains the acknowledgement.

business service property

Key value data pairs used to control the behavior or functionality of business services.

Business Service Property Admin Tool

An EnterpriseOne application for developers and administrators to manage business service property records.

business service property business service group

A classification for business service property at the business service level. This is generally a business service name. A business service level contains one or more business service property groups. Each business service property group may contain zero or more business service property records.

business service property key

A unique name that identifies the business service property globally in the system.

business service property utilities

A utility API used in business service development to access EnterpriseOne business service property data.

business service property value

A value for a business service property.

business service repository

A source management system, for example ClearCase, where business service artifacts and build files are stored. Or, a physical directory in network.

business services server

The physical machine where the business services are located. Business services are run on an application server instance.

business services source file or business service class

One type of business service artifact. A text file with the .java file type written to be compiled by a Java compiler.

business service value object template

The structural representation of a business service value object used in a C-business function.

Business Service Value Object Template Utility

A utility used to create a business service value object template from a business service value object.

business services server artifact

The object to be deployed to the business services server.

correlation data

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

embedded application server instance

A WLS instance started by and running wholly within JDeveloper.

Enterprise Service Bus

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).

exposed method or value object

Published business service source files or parts of published business service source files that are part of the published interface. These are part of the contract with the customer.

HTTP Adapter

A generic set of services that are used to do the basic HTTP operations, such as GET, POST, PUT, DELETE, TRACE, HEAD, and OPTIONS with the provided URL.

internal method or value object

Business service source files or parts of business service source files that are not part of the published interface. These could be private or protected methods. These could be value objects not used in published methods.

JDeveloper Project

An artifact that JDeveloper uses to categorize and compile source files.

JDeveloper Workspace

An artifact that JDeveloper uses to organize project files. It contains one or more project files.

Production Published Business Services Web Service

Published business services web service deployed to a production application server.

published business service

EnterpriseOne service level logic and interface. A classification of a published business service indicating the intention to be exposed to external (non-EnterpriseOne) systems.

published business service identification information

Information about a published business service used to determine relevant authorization records. Published business services + method name, published business services, or *ALL.

published business service web service

Published business services components packaged as J2EE Web Service (namely, a J2EE EAR file that contains business service classes, business service foundation, configuration files, and web service artifacts).

SOA

Abbreviation for Service Oriented Architecture.

softcoding

A coding technique that enables an administrator to manipulate site-specific variables that affect the execution of a given process.

web service softcoding record

An XML document that contains values that are used to configure a web service proxy. This document identifies the endpoint and conditionally includes security information.

web service softcoding template

An XML document that provides the structure for a soft coded record.

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.

Index

A

B

C

D

E

F

H

I

J

L

O

P

R

S

T

U

V

W

X