Oracle® Dynamic Services

User's and Administrator's Guide

Release 9.0.1

June 2001

Part No. A88783-01

Oracle Dynamic Services is a Java-based programmatic framework for incorporating, managing, and deploying Internet services. Oracle Dynamic Services makes it easy for application developers to rapidly incorporate existing services residing in Web sites, local databases, or proprietary systems into their own applications.

ORACLE®

Oracle Dynamic Services User's and Administrator's Guide, Release 9.0.1

Part No. A88783-01

Primary Authors: Alok Srivastava, Marco Carrer, Paul Lin, Wei Qian, Sam Lee, Kan Deng, Cheng Han, Alan Wu, Rod Ward

Contributing Authors: Timothy Chien, Michael Sekurski, Christine Chan, Joseph Meeks, Bill Beauregard, Katherine Oakey, Larry Guros, Yoko Mizuno, Susan Shepard

Contents

7 Service Administration

8 Known Issues and Problems

A Links

B Frequently Asked Questions

C Descriptive Matrix

D Sample Service Packages

E Error Messages

Glossary

Index

List of Examples

List of Figures

List of Tables

Send Us Your Comments

Oracle Dynamic Services User's and Administrator's Guide, Release 9.0.1

Part No. A88783-01

Oracle Corporation welcomes your comments and suggestions on the quality and usefulness of this document. Your input is an important part of the information used for revision.

- Did you find any errors?
- Is the information clearly presented?
- Do you need more information? If so, where?
- Are the examples correct? Do you need more examples?
- What features did you like most?

If you find any errors or have any other suggestions for improvement, please indicate the document title and part number, and the chapter, section, and page number (if available). You can send comments to us in the following ways:

- Electronic mail: nedc-doc_us@oracle.com
- FAX: 603.897.3825 Attn: Oracle Dynamic Services Documentation
- Postal service:
  Oracle Corporation
  Oracle Dynamic Services Documentation
  One Oracle Drive
  Nashua, NH 03062-2804
  USA

If you would like a reply, please give your name, address, telephone number, and (optionally) electronic mail address.

If you have problems with the software, please contact your local Oracle Support Services.

Preface

Oracle Dynamic Services is a Java-based programmatic framework for incorporating, managing, and deploying Internet services. Oracle Dynamic Services makes it easy for application developers to rapidly incorporate existing services residing in Web sites, local databases, or proprietary systems into their own applications.

Audience

This guide is for developers who want to easily and more quickly develop customized, dynamic, Internet service offerings as business opportunities for their customers. An understanding of Oracle9*i*, Java, and XML is required.

Organization

This guide contains the following chapters and appendixes:

Chapter 1	Introduces Oracle Dynamic Services; explains concepts.
Chapter 2	Describes the Oracle Dynamic Services installation.
Chapter 3	Describes Oracle Dynamic Services configuration and how to use Oracle Dynamic Services.
Chapter 4	Describes advanced installation options.
Chapter 5	Describes the Java and PL/SQL Web application development interfaces for accessing the Dynamic Services engine.
Chapter 6	Describes how to build a service.

Chapter 7	Describes service administration tasks.
Chapter 8	Describes known issues and problems with the current release of Oracle Dynamic Services.
Appendix A	Describes some helpful links to W3C specifications.
Appendix B	Describes some frequently asked questions (FAQ).
Appendix C	Describes the descriptive matrix of the schemas and adaptors supplied by Oracle Dynamic Services.
Appendix D	Describes the sample service packages.
Appendix E	Describes Oracle Dynamic Services error messages.
Glossary	Describes the Oracle Dynamic Services terms.

Related Documents

Note: For information added after the release of this guide, see the online README.txt file in your *ORACLE_HOME* directory. Depending on your operating system, this file may be in:

On UNIX systems:

ORACLE_HOME`/ds/doc/README.txt`

On Windows NT systems:

ORACLE_HOME`\ds\doc\README.txt`

See your operating-system specific installation guide for more information.

For the latest documentation, see the Oracle Technology Network Web site:

`http://otn.oracle.com/`

For more information, see the following manuals:

- *Oracle9i XML Reference*
- *PL/SQL User's Guide and Reference*
- *Oracle Internet Directory Administrator's Guide*

- *Oracle9i Java Developer's Guide*
- *Oracle9i Java Stored Procedures Developer's Guide*
- *Oracle9i Enterprise JavaBeans Developer's Guide and Reference*
- *Oracle9i JDBC Developer's Guide and Reference*
- *Oracle9i SQLJ Developer's Guide and Reference*

Conventions

In this guide, Oracle Dynamic Services is sometimes referred to as Dynamic Services.

The following conventions are used in this guide:

Convention	Meaning
. . .	Vertical ellipsis points in an example mean that information not directly related to the example has been omitted.
. . .	Horizontal ellipsis points in statements or commands mean that parts of the statement or command not directly related to the example have been omitted
boldface text	Boldface in text indicates a term defined in the text, the glossary, or in both locations.
< >	Angle brackets enclose user-supplied names.
[ ]	Brackets enclose optional clauses from which you can choose one or none.

Documentation Accessibility

Oracle's goal is to make our products, services, and supporting documentation accessible to the disabled community with good usability. To that end, our documentation includes features that make information available to users of assistive technology. This documentation is available in HTML format, and contains markup to facilitate access by the disabled community. Standards will continue to evolve over time, and Oracle is actively engaged with other market-leading technology vendors to address technical obstacles so that our documentation can be accessible to all of our customers. For additional information, visit the Oracle Accessibility Program Web site at

`http://www.oracle.com/accessibility/`

JAWS, a Windows screen reader, may not always correctly read the code examples in this document. The conventions for writing code require that closing braces should appear on an otherwise empty line; however, JAWS may not always read a line of text that consists solely of a bracket or brace.

1
Introduction

As a feature of Oracle9*i*, Oracle Dynamic Services is a Java-based programmatic framework for incorporating, managing, and deploying Internet and Intranet services. Using the Internet as the information source, Oracle Dynamic Services makes it easy for application developers to rapidly incorporate valuable services from Web sites, local databases, and proprietary systems into their applications.

For example, an online financial portfolio can use Oracle Dynamic Services to integrate Internet financial services, such as stock quotes and exchange rates, from different resource providers to calculate the current value of a portfolio in foreign currency. Oracle Dynamic Services is designed to handle dynamic business models with no degradation in the quality of service. Business opportunities can be maximized because this framework permits customized service delivery for flexible application development.

Using the simple, yet flexible framework of Oracle Dynamic Services, application developers can significantly shorten the development cycle for developing applications and increase quality of service by selecting the very best sources. With the Internet becoming the source of choice to compose, deploy, access, and manage business information through service offerings, Oracle Dynamic Services provides the best framework to dynamically manage and customize these Internet services.

Table 1–1 summarizes the tasks or roles of people or organizations in the Dynamic Services framework; the roles of these people and organizations are described later in this section.

Table 1–1 Summary of People or Organizations and Their Tasks or Roles in the Oracle Dynamic Services Framework

People or Organizations	Tasks or Roles
Service provider (business partner and application developer)	Owns business information content or services to offer Provides the specification for accessing services Provides the content for a service Designs the service package Registers the service package
Service consumer application	Incorporates services into applications in order to deploy them as useful, customized services Designs the flow of the services
Service administrator	Grants service access to service consumer applications Registers the service package Manages the Oracle Dynamic Services registry Manages service consumer applications Performs tuning of the Oracle Dynamic Services engine

1.1 Application Scenarios

Businesses and application developers face a host of business problems and technical challenges in providing information and integrating it into dynamic applications for the Internet or for corporate Intranets.

1.1.1 Business Problems or Technical Challenges

To integrate Internet services or Intranet services into dynamic applications, businesses must be able to:

- Access sources using protocols or APIs
- Manage sources that have different types of sessions with different protocols, have multiple content formats, and do not have guaranteed access
- Deliver specially formatted content to both Web and wireless devices

To access a variety of information sources, businesses must do the following:

- Develop custom code to manage data transformations among different protocols

- Develop custom servers to use these resources
- Implement security, scalability, and performance with every server and then manage them all

In addition, due to the extensive code customization required, businesses rarely can reuse these custom servers.

To manage these information resources, businesses must manage sessions differently for different protocols such as cookies for HTTP. They must be able to handle various content structures, such as DB result sets, XML, Java objects, and so forth; and, they must develop custom solutions for failover service aggregation, caching, and so forth.

To deliver content, businesses must render application results into multiple formats, such as HTML, XML, and so forth. Due to the difficulty involved, they often must utilize consultants to integrate their applications. They must continually change code to adapt to any changes because nothing is configurable; and they face great challenges in being able to scale their applications as their customer base expands.

In summary, it is currently very expensive and an extremely complex operation to develop applications for customers because of the extraordinary number and variety of technical issues involved.

1.1.2 Oracle Dynamic Services Solutions

Application developers who develop e-business, wireless, or portal applications must be able to easily and rapidly aggregate service offerings from business partners and application developers (often referred to as service providers) and provide a single service visible to customers (see Figure 1–1). Application developers, who build their service offerings upon a sound architecture, can quickly develop a collection of services that are easily maintained and managed, and rapidly deployed to meet changing business needs.

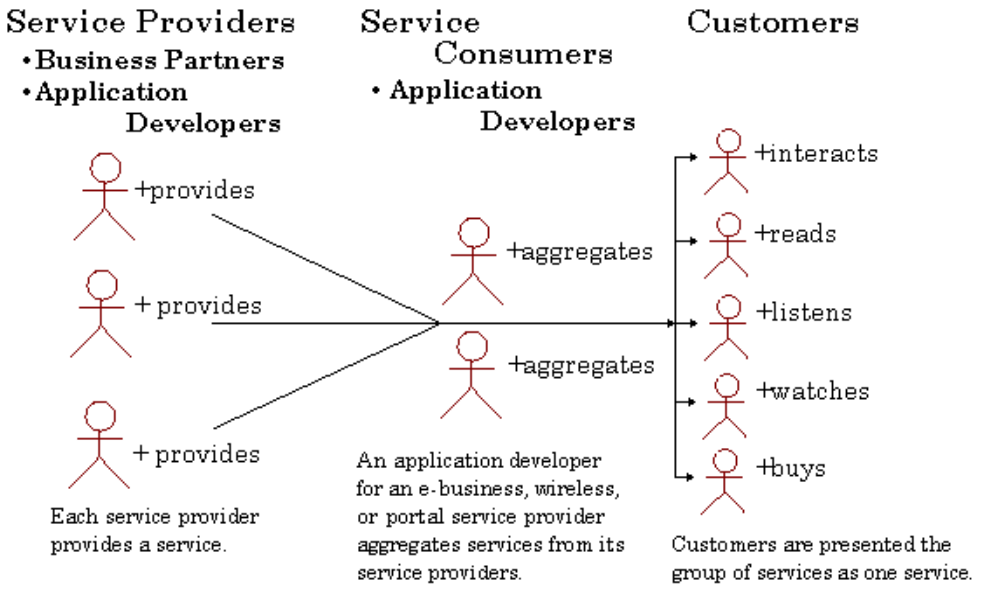
Application developers can use Oracle Dynamic Services to create customized delivery of services with the following benefits:

- Separating of application logic from service access for improved application development and easier maintenance
- Modifying available services to solve business problems, thereby reducing the cost and time needed to develop a new application
- Making development of value-added services or applications easier and faster by using services from multiple resource providers

- Enabling improvement of the incorporated services without affecting existing applications

Service consumers, such as an application developer for an e-business, wireless, or portal service provider, provides one aggregate set of services to customers as shown in Figure 1–1, based on specifications provided by the service providers.

Figure 1–1 Application Developers Aggregate Services for Customers



Oracle Dynamic Services can be used in a number of scenarios including wireless, portal, and e-business applications, where services are integrated with little incremental development cost.

1.1.2.1 Wireless Service Developers

Wireless service developers can use Oracle Dynamic Services to incorporate various useful services. Applications can be built for wireless (handheld) devices to prompt messages, advertisements, or special services based on the geographical location of the user with the handheld device. For example, as a user enters a particular geographic area, an application can prompt specific information to the user's device, highlighting specific local business offerings, such as store sales promotions,

a significant weather event, local entertainment, and so forth. Using Oracle Dynamic Services, application developers can aggregate multiple services into a single service, and provide a great breadth of information to customers.

1.1.2.2 Portal Service Developers

Portal service developers, who can deliver a richer, broader, and more dynamic array of information and services to users for specific geographical areas, become more powerful in attracting users to their service offerings. Using Oracle Dynamic Services, portal developers can build applications that contain services to perform specific tasks for the user. These applications can, for example, send confirmation notices when each task is completed. Using a portal search engine, local restaurants of interest can be listed, a reservation made, and a confirmation notice returned along with directions to the restaurant from a specific point of interest. As the success of these kinds of services grows, so does the use of the portal by its customers.

1.1.2.3 e-Business Service Application Developers

As e-businesses grow, application developers can use Oracle Dynamic Services to integrate different services to help its customers through a series of related transactions, such as buying a house. Using an online real estate service, a prospective buyer can locate houses of interest, visit each house through a virtual tour, ask questions, and decide upon a small set of those houses to visit in person. This same buyer can also locate several mortgage lenders, fill out an online mortgage loan application, schedule home inspections, loan closures, movers, and so forth. Every possible real estate service can be used as needed to make the house-buying experience as enjoyable and easy as possible for the home buyer.

Using the *service triggering* capability of Oracle Dynamic Services, as one task is completed, the next set of related tasks is scheduled, so in turn, each task leads directly to additional related tasks. Application developers for the online real estate broker can write an entire e-business application using Oracle Dynamic Services. The real estate broker needs only to cultivate and manage the relationships among the various service providers.

1.2 Overview of Concepts

Table 1–2 describes the primary components that comprise Oracle Dynamic Services.

Table 1–2 Dynamic Services Components and Their Functions

Components	Functions
Service consumer application	Acts as a client of the Dynamic Service engine, writes applications using this framework.
Dynamic Services client library	Handles communication between the service consumer application and the Dynamic Services engine. Connects an application with the Dynamic Services engine by opening a connection to it, in a fashion similar to opening a JDBC connection. There are multiple connection drivers available with Dynamic Services that allow different connection paths from applications to the Dynamic Services engine. Applications must register the desired driver and then operate with the returned connection. (See Section 5.1.3 for more information.)
Service package	Contains the information necessary to model a resource as a service component deployable in the Oracle Dynamic Services framework. Contains, in its simplest form, a bundle of files modeled as a local directory. Contains, in its compound form, an additional file, a jar file, containing all Java classes and stylesheets needed by the compound service.
Service registry	Maintains the service package information of registered services that enables Dynamic Services engines to set up and execute a service, and access distributed sources from service providers.
Application profile registry	Maintains service consumer application information about the identity of service consumer applications and their properties. A service consumer application must be registered in the application profile registry.

Table 1–2 Dynamic Services Components and Their Functions (Cont.)

Components	Functions
Dynamic Services engine	Accepts service requests from client applications and does the following tasks: ■ Performs post-processing of service requests to produce the input required by the service (input adaptor) ■ Determines how the service needs to be executed and sets up the service execution environment (execution adaptor) ■ Issues service execution requests to the service providers by transforming the standard service request to the input needed by the service following the underlying protocol (protocol adaptor) Receives the service response from the service providers and does the following task: ■ Transforms the service response for the client and returns it to the caller (output adaptor) Can execute services in synchronous as well as asynchronous mode, depending upon the client application setup.
Service administrator	Uses an extended version of the Dynamic Services client library for communicating with the Dynamic Services engine. Includes an administration shell (DSAdmin utility) and a Web-based administration utility that are both part of the Dynamic Services engine to manage that engine and all its components.

Figure 1–2 shows the Oracle Dynamic Services architecture. Service providers (business partners and application developers) provide services that service administrators register in the service registry using the DSAdmin utility. Application developers create applications using application profiles that service administrators register in the application profile registry. The registry is an Oracle Internet Directory (OID) Lightweight Directory Access Protocol (LDAP) server whose contents are also mirrored in the Oracle9*i* database for performance optimization. The Dynamic Services Java engine, depending upon the configuration, can reside either inside or outside Oracle9*i*. Dynamic Services does the following:

- Exposes PL/SQL interfaces to run the Oracle Dynamic Services engine within Oracle9*i* JVM (see Figure 1–5)
- Exposes Java interfaces when it runs on a local machine hosting the application (thick client library) (see Figure 1–4)

- Acts as a middle-tier Java engine behind a Java servlet with the application using a Dynamic Services thin client library (see Figure 1–6)

Figure 1–2 Oracle Dynamic Services Architecture

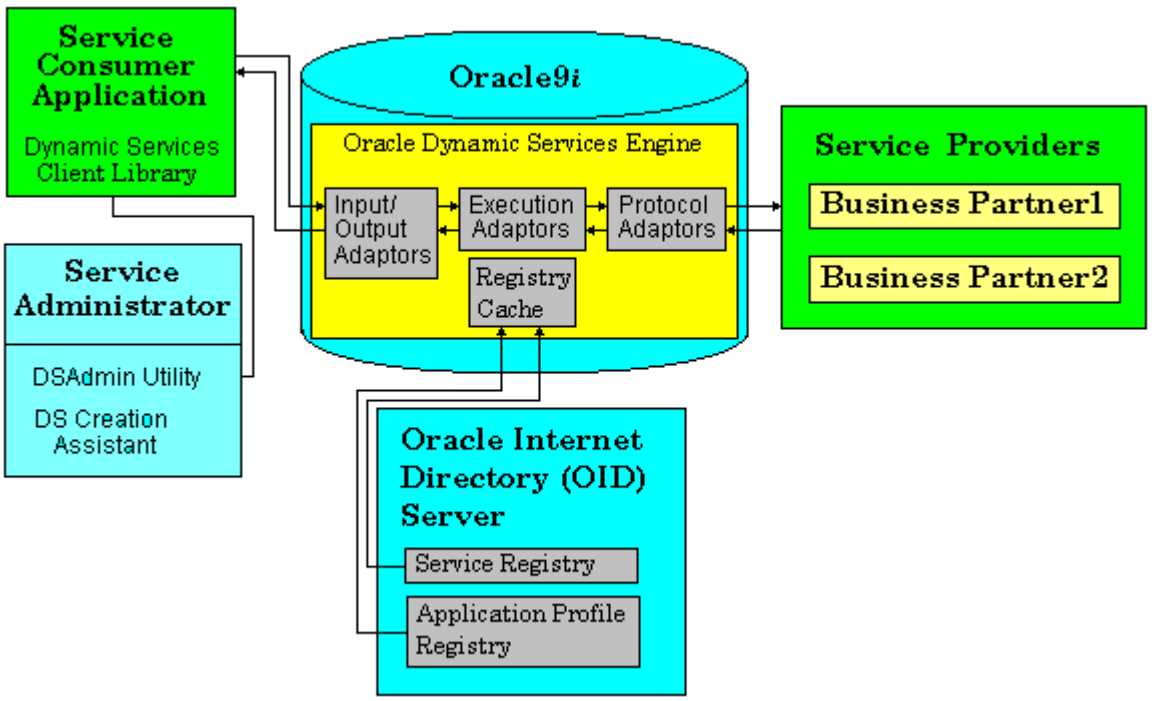
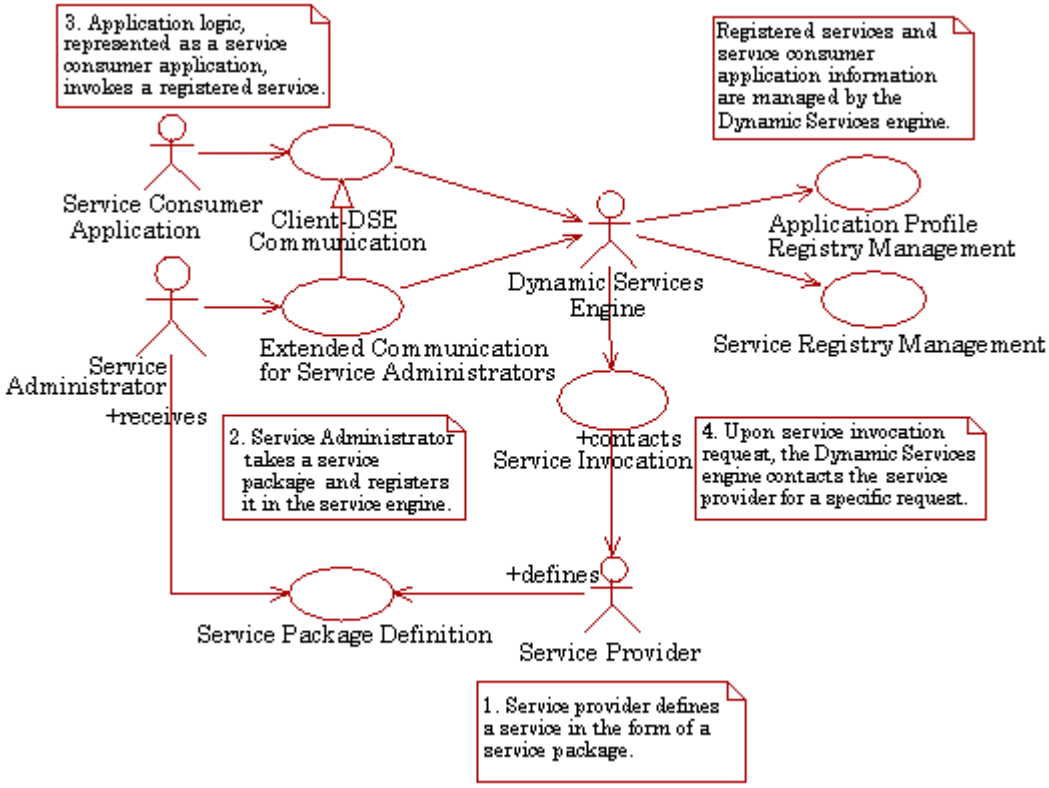


Figure 1–3 shows the major components of Oracle Dynamic Services and the roles of people and organizations in the Dynamic Services framework. These major components and roles begin with the definition of a service package. Both service providers and service consumer applications can define the service package depending on their business relationship. The service administrator takes the service package and registers it in the Dynamic Services engine. Registered services and applications are managed by the Dynamic Services engine. Next, application logic within an application invokes a registered service. Upon the service invocation request, the Dynamic Services engine then contacts the service provider for the specific request.

Figure 1–3 Roles in the Oracle Dynamic Services Framework



1.2.1 Service Provider

Service providers provide the content and transformations for a service. Service developers and service administrators define the service and load it into the service registry so that the service becomes available to service consumer applications. Service consumer applications can combine many of these services to create value-added services or applications. Service developers need to know the requirements of the service providers for access and authentication in order to create a service in the Dynamic Services framework. Service developers can also specify caching policy, failover, and so forth, to further improve scalability and

reliability. The Dynamic Services engine contacts the service providers during service execution according to the specification provided in the registered service package.

1.2.2 Service Registry

The service registry is the storage place for the service package. A service package enables Dynamic Services engines to set up and execute a service, and access distributed information sources from service providers. Service consumer applications can use the client library to perform lookup operations on the service registry. Service administrators can perform updates on the registry without affecting client applications. This feature simplifies the client applications.

1.2.3 Application Profile Registry

The application profile registry is the storage place for the service consumer application attributes. It holds information about the identity of service consumer applications and their properties. A service consumer application must be registered to the Oracle Dynamic Services engine. Before a service consumer application is registered, it must be associated with a database user who has been granted the connect privilege and been granted the DSUSER_ROLE privilege. Then, the named service consumer application must be granted service execution privileges for a service before it can access the named service.

1.2.4 Service Administrator

The service administrator is responsible for managing the Dynamic Services engine and all of its components. The service administrator monitors service failover, manages caching policy, schedules services, and also registers and unregisters services and service consumer applications. The service administrator can listen to the events raised within the Dynamic Services engine to monitor, trace, profile, view service execution, and view service session data. The service administrator also specifies deployment options for services and controls service access to the service consumer applications. The service administrator performs engine performance monitoring, service log reviewing, and so forth.

1.2.5 Service Consumer Application

A service consumer application acts as a client of the Dynamic Service engine. Through the Dynamic Services client API, service consumer applications acquire handles on the services it wants to execute, submits service execution requests, and

collects the responses. Service consumer applications need not be aware of the communication protocol used by the Dynamic Services client library and the Dynamic Services engine. The communication protocol is abstracted by the Dynamic Services framework. Service administrators are also unaware of the service providers and other management infrastructure supporting the service execution. This abstraction has been built into the Dynamic Services framework to keep the client applications simple and less vulnerable to the changing business conditions and the changing technical environment that supports their applications.

1.2.6 Dynamic Services Engine

The core of the Dynamic Services framework is the Dynamic Services engine. The Dynamic Services engine is a multithreaded Java engine, which accepts requests from the client applications. The Dynamic Services engine can execute services in synchronous as well as asynchronous modes, depending upon the client application setup. Once a request is received by the engine, the engine determines how the service needs to be executed, sets up the execution environment, and issues execution requests to the service providers. Upon receiving the response from the service providers, the engine transforms the response for the client and returns it to the caller.

1.2.7 Services as Application Components

One of the premier advantages of using Oracle Dynamic Services is the ability to use services as application components. Service administrators can easily change a service provider, because as a service, their access to service consumer applications is easily managed. Application components can be easily aggregated to offer a service composed of many services. For example, an application can offer failover service aggregation among a group of related services should a specific service become unavailable. An application can offer a specific set of services based on certain business conditions, and so forth. Furthermore, specific application components can be easily tailored to deliver content in a format suitable for the device an end user is using. Because these options are available as application components, applications can be rapidly developed and deployed as well as modified to fit changing business needs. The ability to easily build, maintain, and manage a collection of application components for rapid deployment is what Oracle Dynamic Services offers to application developers.

1.2.8 Communication Between the Service Consumer Application and the Dynamic Services Engine

The client library is responsible for handling the communication between the service consumer application and the Dynamic Services engine. The communication is performed as synchronous or asynchronous messages between the client library and the Dynamic Services engine. Service consumer applications can communicate with any available Dynamic Services engine provided they are authorized to use that particular instance. Once connected, service consumer applications have the access and privileges that service administrators assign to them. These features allow distributed client access to a large number of Dynamic Services engines, and can be used to implement client failovers or load balancing.

1.2.9 Communication Between the Service Administrator and the Dynamic Services Engine

The service administrator interfaces with the Dynamic Services engine through the administrator tools. The administrator tools use an extended version of the client library to communicate with the Dynamic Services engine. The Oracle Dynamic Services administrative shell, shipped with the Dynamic Services engine, is an example of these tools. This is an interactive, scriptable, easy-to-use command-line shell that includes online help. Some of the features of the shell are also available from a Web-based administration utility, which is shipped with the Dynamic Services engine.

1.3 Dynamic Services Implementation Overview

Oracle Dynamic Services has three possible deployment modes:

- Java deployment view (see Section 1.3.1)
- PL/SQL deployment view (see Section 1.3.2)
- Java (HTTP/Java Messaging Services (JMS)) Deployment view (see Section 1.3.3)

The following is a brief description of the underlying technologies of the high-level components for each implementation.

Dynamic Services Engine

The Dynamic Services engine can be deployed as any of the following three engine types:

- A Java engine running on the machine hosting the application (thick client library) (see Figure 1–4)
- A middle-tier Java engine behind a Java servlet (see Figure 1–6)
- A Java engine running within Oracle9*i* JVM (see Figure 1–5)

Different options can be selected by service consumer applications based upon their application needs. A unique feature of the Dynamic Services framework is that service consumer applications can switch from one environment to another without recompiling or even restarting their applications. This gives the service consumer application added flexibility to try out various options, to see which best fit their applications.

Dynamic Services Service and Application Profile Registries

The service registry and application profile registry are deployed as directories in the Oracle Internet Directory (OID) server. The access control list of OID is used for access control, allowing service administrators to choose the services visible to a particular service consumer application. Managing services and service consumer applications in OID allows multiple instances of Dynamic Services engines to work in a synchronized fashion, giving an open, scalable option to service consumer applications. For performance reasons, the registry data is cached in an Oracle9*i* instance accessed by the Oracle Dynamic Services engine at service execution time. This cache can be synchronized automatically at the start of the Dynamic Services engine, or service administrators can synchronize it through their console, as required.

Communication Between Service Consumer Applications and the Dynamic Services Engine

The communication between the Dynamic Services engine and the service consumer applications is abstracted by the Dynamic Services client library. By registering a Dynamic Services driver, a service consumer application can dynamically change the underlying communication protocol used by the client library to communicate with the Dynamic Services engine. Supported communication protocols include HTTP (see Figure 1–6), AQ/JMS (see Figure 1–6), and direct Java access (see Figure 1–4). Service consumer applications have complete control over the drivers they choose within their programming framework, and they can switch to any driver. Service consumer applications can use multiple drivers to talk to multiple Dynamic Services engines, at the same time, if required.

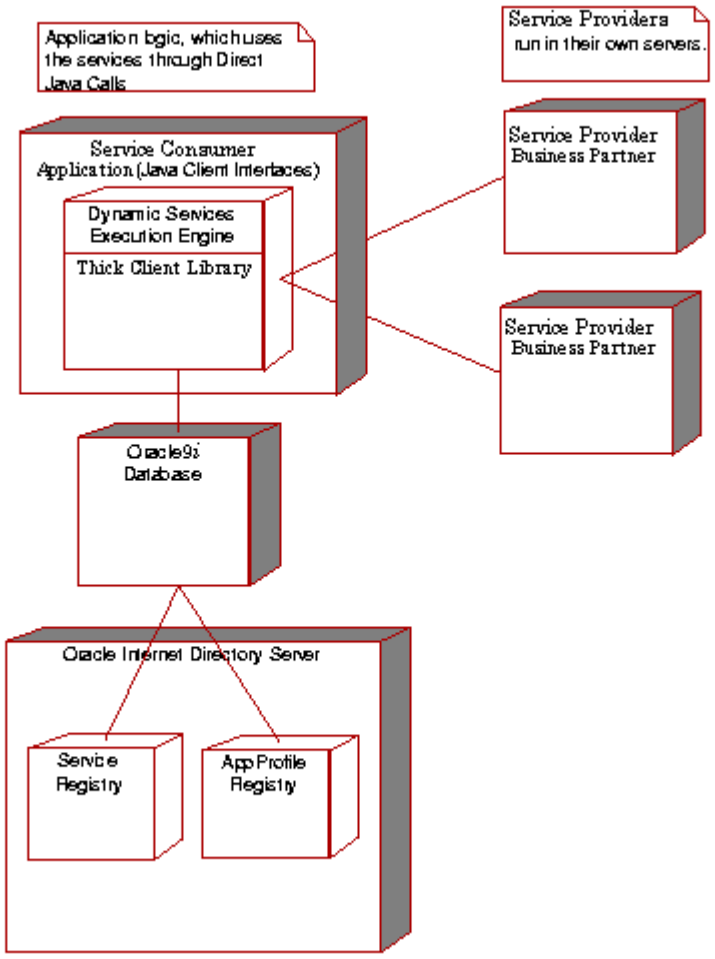
Service consumer applications can access services through different paths depending upon their Dynamic Services engine deployment. The Dynamic Services engine allows access to the services in PL/SQL and Java for programming purposes.

1.3.1 Java Deployment View

Figure 1–4 shows a basic Java deployment view of the Oracle Dynamic Services framework. The Oracle9*i* database serves as a registry cache, communicating with the OID Lightweight Directory Access Protocol (LDAP) server hosting the registries. The service consumer application contains application logic that uses the services through direct Java calls.

In this case, the service consumer application uses the Dynamic Services thick Java client library, which contains the Dynamic Services execution engine. Service providers run in their own servers.

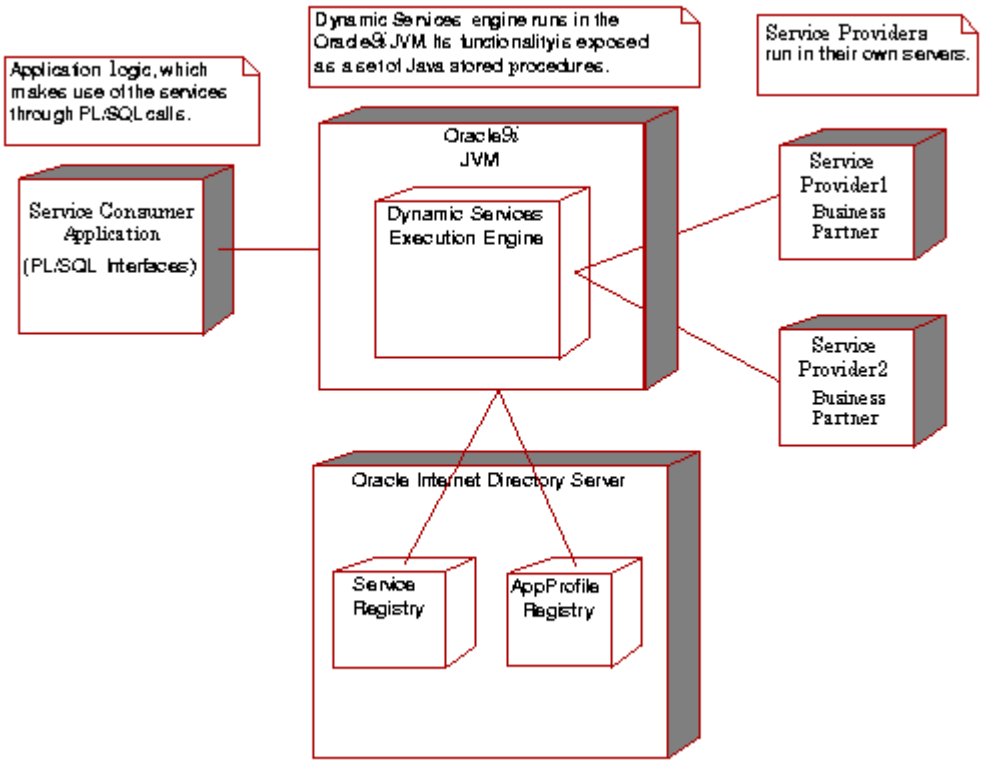
Figure 1–4 Java Deployment View of the Oracle Dynamic Services Framework



1.3.2 PL/SQL Deployment View

Figure 1–5 shows a PL/SQL deployment view of the Oracle Dynamic Services framework. The Dynamic Services engine runs in the Oracle9*i* JVM, with its functions exposed as a set of Java stored procedures. The Oracle9*i* database serves as a registry cache, communicating with the Oracle Internet Directory LDAP server hosting the registries. The service consumer application contains application logic, which makes use of the services through PL/SQL calls. Service providers run in their own servers.

Figure 1–5 PL/SQL Deployment View of the Oracle Dynamic Services Framework



1.3.3 Java (HTTP/Java Messaging Services (JMS)) Deployment View

Figure 1–6 shows a Java (HTTP / JMS) deployment view of the Oracle Dynamic Services framework. The Dynamic Services engine runs in a Dynamic Services gateway (middle tier) that supports HTTP, HTTPS, and JMS as communication protocols. The Oracle9*i* database serves as a registry cache, communicating with the Oracle Internet Directory LDAP server hosting the registries. The service consumer application contains application logic, which makes use of the services through the Dynamic Services thin Java client library, and can execute services remotely in other systems.

In this case, service execution requests are forwarded to the Dynamic Services gateway, which executes the service and returns the response. The communication between the service consumer application and the gateway is handled by the Dynamic Services thin Java client library.

Figure 1–6 Java (HTTP/JMS) Deployment View of the Oracle Dynamic Services Framework

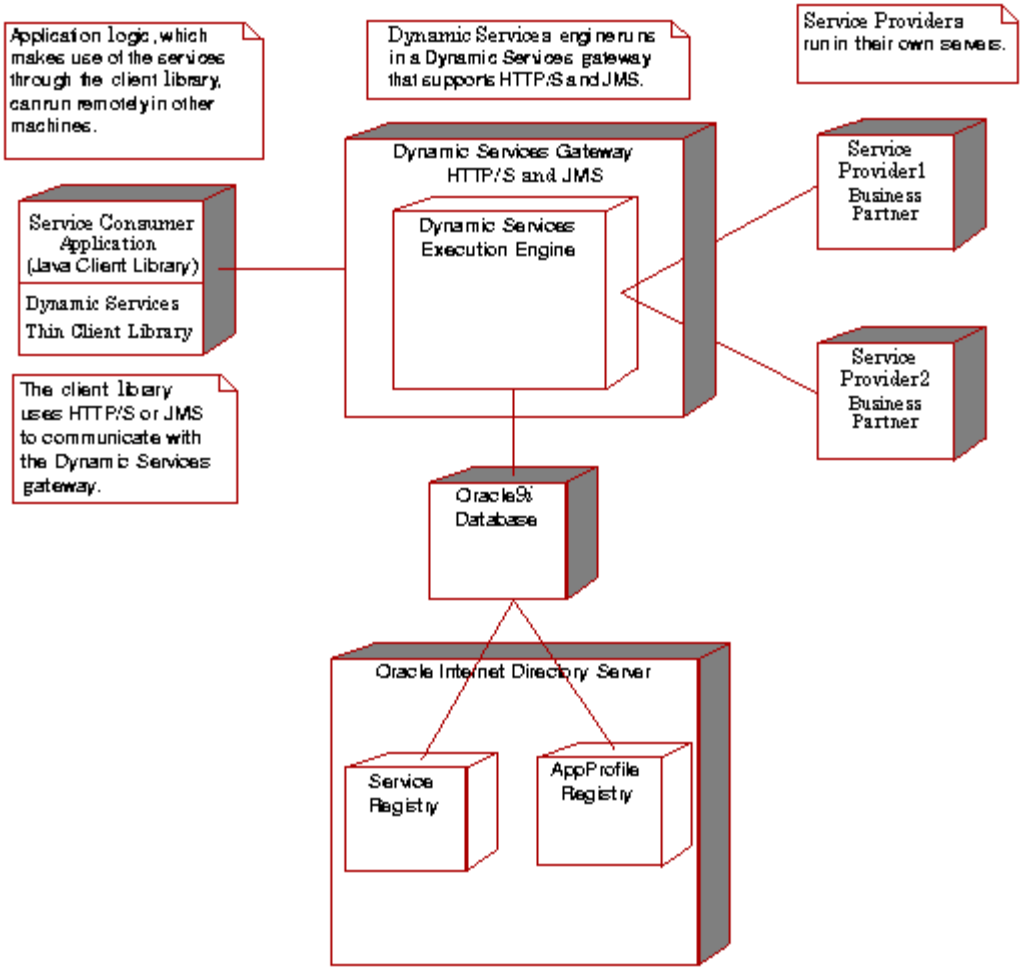
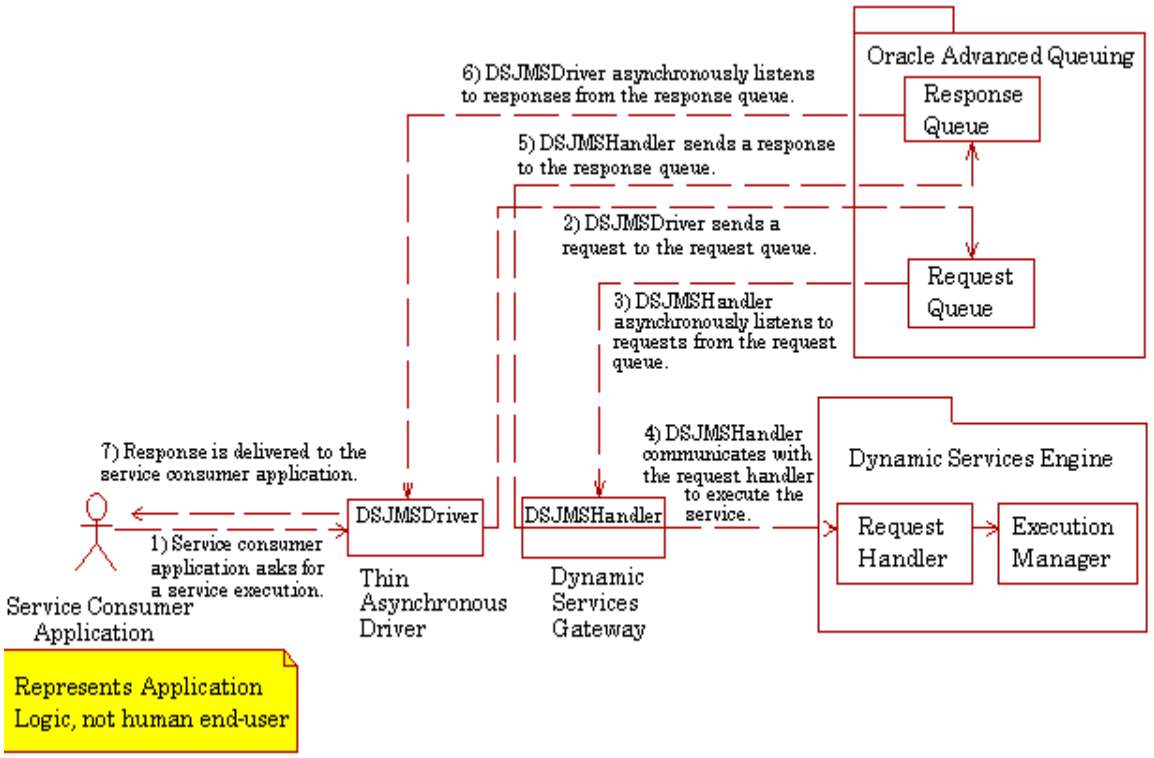


Figure 1–7 shows the asynchronous deployment communication (JMS) that occurs when the DSJMSDriver allows for asynchronous access to services using a Dynamic Services gateway in the form of a JMS daemon. The mode of operations with this

driver lets it submit requests asynchronously to an AQ/JMS queue in a remote database. The driver assumes the existence of this JMS daemon that listens asynchronously to the same queue where requests are being submitted. The daemon takes on the role of the Dynamic Services engine and processes the request, generates a response, and submits that response into another queue that the DSJMSDriver listens to asynchronously. On the service consumer application side, therefore, listeners can be registered to be informed when the response is returned.

Figure 1–7 Asynchronous Deployment Communication (JMS)



1.4 Using Multiple Dynamic Services Engines

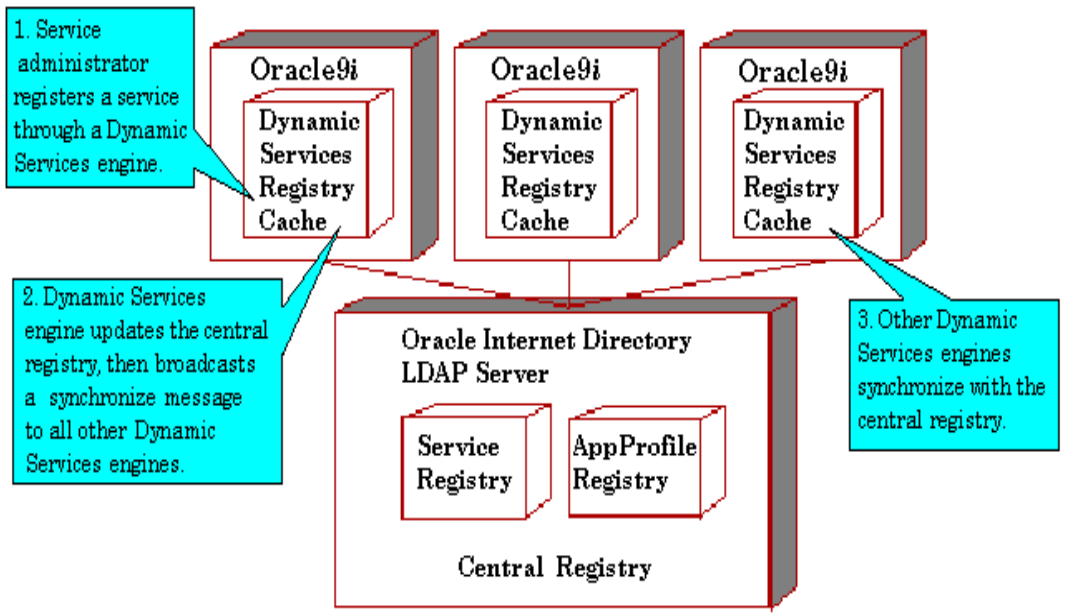
To increase scalability, you can install multiple Dynamic Services engines that communicate with a central master Lightweight Directory Access Protocol (LDAP)

registry (see Figure 1–8). See Section 4.5 for installation and configuration information for setting up LDAP with OID and configuring the Dynamic Services registry to use LDAP.

The basic steps for using LDAP as a central master registry are as follows:

1. The service administrator registers a service through one Dynamic Services engine.
2. This Dynamic Services engine updates the central registry, then broadcasts a synchronize message to all other instances of the Dynamic Services engines.
3. All other instances of Dynamic Services engines synchronize their registry cache with the central registry.

Figure 1–8 Using Multiple-Instance Deployment of Oracle Dynamic Services Engines



1.5 How to Get Started with Oracle Dynamic Services

The remaining chapters in this guide begin by guiding you through a basic installation and configuration of Oracle Dynamic Services (Chapter 2), then

showing you how to get started by configuring and running the DSAdmin utility and registering a new service, browsing registered services, and executing a registered service (Chapter 3).

Advanced topics are discussed in the remaining chapters, guiding you through advanced installation and configuration options (Chapter 4), describing how to use the Java and PL/SQL Web application development interfaces (Chapter 5), showing you the process of service development (Chapter 6), and finally describing service administration tasks (Chapter 7).

2
Installation

This chapter describes the basic installation and configuration of Oracle Dynamic Services, which is the Java deployment view described in Section 1.3.1.

The installation requires that the person installing this software have sysadmin privileges, is a database administrator, and has application development skills.

After you have completed the installation steps described in this chapter, you will have an understanding of the system requirements needed to install Dynamic Services and will have installed the DSSYS schema. You *must* continue to Chapter 3 to configure the DSAdmin utility, and use this utility to register and execute a service to ensure that Dynamic Services is properly installed and running.

To configure and use other deployment views of Dynamic Services, such as HTTP / HTTPS, PL / SQL, or HTTP / Java Messaging Services (JMS), or to set up LDAP as a central master registry, see the appropriate sections described in Chapter 4. Chapter 4 also describes other advanced features such as enabling persistent auditing, manually fine-tuning Dynamic Services properties, and installing the management console.

Note: The version requirements for Oracle9*i* Standard Edition or Enterprise Edition refer to Release 1 (9.0.1).

2.1 System Requirements

The following are the system requirements:

- Oracle release: Oracle9*i* Standard Edition or Enterprise Edition Release 1 (9.0.1) is required to install and use Oracle Dynamic Services.

> **Note:** <*ORACLE_HOME*> is referred to as the installation directory of the Oracle9*i* Release 1 (9.0.1) distribution.

- Oracle Dynamic Services supports Oracle9*i* Standard Edition and Enterprise Edition running JDK 1.2.2.
- Java version: JDK 1.2.2 or later (Java2) distribution.

> **Note:** <*JAVA2_HOME*> is the installation directory of the JDK 1.2.2 or later distribution.

Ensure you have a full installation of Oracle9*i* Release 1 (9.0.1) (a full installation in this case includes a typical Oracle9*i* Release 1 (9.0.1) installation. Follow Oracle9*i* installation instructions to complete a full installation.

> **Important:** In Section 2.2, directory paths often show only the UNIX path "/" specification. If you are installing on a Windows NT system, the path specification is "\" and you must make this change, as needed, for the installation to be successful.

2.2 Dynamic Services Distribution

For release 9.0.1, Oracle Dynamic Services is installed using the Oracle Universal Installer into the *ds* directory within your <*ORACLE_HOME*> directory. The distribution contains the subdirectories shown in Table 2–1.

Table 2–1 Oracle Dynamic Services ds Directory Contents

Subdirectories	Description
bin	Contains the DSAdmin command-line utility for registering or unregistering a service, and running the test service executions.
demo/consumer	Contains the sample client code for the service consumer.
demo/dsadmin	Contains the sample Oracle Dynamic Services script files.
demo/services	Contains the sample service packages.

Table 2–1 Oracle Dynamic Services ds Directory Contents (Cont.)

Subdirectories	Description
doc	Contains the documentation about Dynamic Services including the README.txt file, the Oracle Dynamic Services FAQ file (dsfaq.txt), and the JavaDoc API (apidoc.zip), which contains those classes that are necessary for service consumers and service developers to build and run services.
etc	Contains miscellaneous configuration files.
etc/Apache_JServ	Contains a sample servlet zone configuration file.
etc/dsadmin	Contains a collection of system scripts and a properties file.
etc/services	Contains the service packages needed for the monitor services.
etc/xsd	Contains the XML schema for the service descriptor and supplied adaptors.
jsp/management	Contains the Management Console application.
ldif	Contains the Lightweight Directory Access Protocol (LDAP) schema definitions for the Oracle Internet Directory (OID) registry.
lib	Contains the jar file of the Dynamic Services code.
sql	Contains SQL scripts for installing in Oracle9*i* the support necessary for Oracle Dynamic Services.
logs	Contains log files used by JMS daemon.

2.3 Installing the DSSYS Schema

The DSSYS schema SQL scripts do the following:

- Create the tablespaces for the DSSYS schema.
- Connect to the schema as DSSYS/<*DSSYS-password*>.
- Create all object and table definitions.
- Install the Dynamic Services registry.
- Initialize the user profile registry.
- Create a table for caching service responses.
- Install the Dynamic Services cache manager package.
- Create the Dynamic Services properties table.

- Load the Dynamic Services properties package.
- Initialize queues and events.
- Create the Dynamic Services user roles.

To install the DSSYS schema, perform the following tasks:

1. Using a command-line shell, change the current directory to the directory where the DSSYS schema installation script is located, shown as follows:

   On UNIX systems:

   ```
   cd <ORACLE_HOME>/ds/sql
   ```

   On Windows NT systems:

   ```
   cd <ORACLE_HOME>\ds\sql
   ```

2. Connect as SYS to the Oracle9*i* 9.0.1 instance using SQL*Plus as follows:

   ```
   sqlplus sys/<sys-password>
   ```

   The DSSYS schema installation script assumes the connected SYS user has the required privileges to create users.

   Note: The dsinstall.sql script invokes a dssys_ts_init.sql script responsible for the creation of tablespaces that are needed by the DSSYS schema. Review the dssys_ts_init.sql script and customize it before running the dsinstall.sql script. For example, you may want to store these tablespaces on another disk.

   ---

3. Run the dsinstall.sql script by issuing the following command in SQL*Plus:

   ```
   SQL> @dsinstall.sql
   ```

   The install script creates a log file of its execution so that it can be checked for errors. The log file is created within the same directory and is named dsinstall.log. To check if the installation had errors, run the following file at the UNIX prompt:`<ORACLE_HOME>bin/showerrors` (`showerrors.bat` on Windows NT) as follows:

   On UNIX systems:

   ```
   <ORACLE_HOME>/ds/bin/showerrors
   ```

On Windows NT systems:

```
<ORACLE_HOME>\ds\bin\showerrors.bat
```

The `showerrors` command only reports failure errors or installation errors. If there are none shown, then there are no errors to report.

Note: The `bin/showerrors` directory path on UNIX systems or the `bin\showerrors` directory path on Windows NT systems is relative to the Oracle Dynamic Services installation home. On UNIX systems, you must edit the `bin/showerrors` file, or on Windows NT, you must edit the `bin\showerrors.bat` file and enter the correct location for your <*ORACLE_HOME*> location.

---

4. To verify the installation of the schema, exit SQL*Plus and try to reconnect to the database as the DSSYS user by issuing the following command:

   ```
   sqlplus dssys/<dssys-password>
   ```

Note: The default password for user DSSYS after installation is DSSYS. For security reasons, you should change the default password and modify the appropriate files in the installation.

2.4 Dynamic Services Configuration

After the installation is complete, you *must* first configure and run the DSAdmin utility before you can register and execute services, which is the test that your Dynamic Services installation and DSAdmin utility configuration is working properly. To configure the DSAdmin utility and register and execute a service, you *must* continue to Chapter 3.

Important: You *cannot* run any Dynamic Services service registration scripts until you first configure and run the DSAdmin utility described in Section 3.1.2 and Section 3.1.3. Then, to test the DSAdmin configuration, follow the steps described in Section 3.2, Section 3.3, and Section 3.4.

3
Configuration

To get started with Oracle Dynamic Services, you must first configure and run the DSAdmin utility. Then you can use the DSAdmin utility to register a new service, browse through your list of registered services, and finally, execute a registered service. This chapter describes each of these topics.

3.1 Configuring and Running the DSAdmin Utility

To verify a successful installation, use the DSAdmin command-line utility (`dsadmin` on UNIX systems or `dsadmin.bat` on Windows NT).

Note: For UNIX and Windows NT, you must first edit the `dsadmin` or `dsadmin.bat` file to specify the correct *<ORACLE_HOME>* before using the DSAdmin utility. On UNIX systems, the `dsadmin` file is located in the following directory:

<ORACLE_HOME>`/ds/bin/`

On Windows NT systems, the `dsadmin.bat` file is located in the following directory:

<ORACLE_HOME>`\ds\bin\`

The DSAdmin utility allows command-line interactions with the Oracle Dynamic Services engine and lets you perform common operations, such as service registration, service unregistration, and service execution testing.

3.1.1 Configuring Dynamic Services Proxy Settings

In order to connect to services located outside of a firewall for testing the sample service, you must first configure the Dynamic Services proxy settings. To do this, you must run SQL*Plus and connect as dssys user and enter your <dssys-password> and execute three procedures as follows:

```
sqlplus dssys/<dssys-password>
set serveroutput on
exec ds_properties.show()
exec ds_properties.setProperty('proxyHost','<www-your complete proxy name>')
exec ds_properties.setProperty('proxySet','true')
```

For more information about setting these Dynamic Services properties, see Section 4.6.

3.1.2 Configuring the DSAdmin Utility

Before you run the DSAdmin utility, you must configure its configuration XML file, `DSAdminConfig.xml`, located in the following directory:

On UNIX systems:

`$<ORACLE_HOME>/ds/etc/dsadmin/`

On Windows NT systems:

`$<ORACLE_HOME>\ds\etc\dsadmin\`

Note that this is the default path where the DSAdmin utility expects to find its configuration file.

1. Open the `DSAdminConfig.xml` file in an editor.

   With the basic Dynamic Services installation, only connections using the DSDirectDriver driver can be used. Therefore, the only element you need to change is DS_URL for connections that use the Direct driver.

2. Change the DS_URL element to point to your database instance that hosts Oracle Dynamic Services for the connection descriptor with the name "Direct," shown as follows:

```
<DS_CONNECTION_DESCRIPTOR name="Direct">
   <annotation>
     -| For Nickname "Direct":
      | These are specifications of the Direct Driver class
     +| that will be used as well as the URL to be used with it
   </annotation>
```

```
   <DS_DRIVER>oracle.ds.driver.DSDirectDriver</DS_DRIVER>
   <DS_URL>jdbc:oracle:thin:@<your-host-name>:<your-port-number>:<your-SID></DS_URL>
</DS_CONNECTION_DESCRIPTOR>
```

You can also have your own configuration file and point to it by running the command shown in Example 3–1 (the -c option and additional options are described later in this section).

Example 3–1 Configure the DSAdmin Utility

On UNIX systems:

```
<ORACLE_HOME>/ds/bin/dsadmin -c <your config file>
```

On Windows NT systems:

```
<ORACLE_HOME>\ds\bin\dsadmin.bat -c <your config file>
```

The configuration file conforms to the specifications of an XML document containing elements and values. The specific elements that you can configure in the file are described in Table 3–1.

Table 3–1 DSAdmin Utility Configuration Elements

Element	Description
DS_ADMIN_CONFIG	The root element of the DSAdmin utility configuration document.
DS_CONNECTION_ DESCRIPTORS	The connection descriptor section. Contains descriptions of the connection nickname, driver class specification, and associated URL.
DS_CONNECTION_ DESCRIPTOR	The connection descriptor. Contains the name attribute describing the nickname for the connection that is used to open Dynamic Services connections to be used throughout the lifetime of the DSAdmin utility.
DS_DRIVER	The driver class name. This driver class will be loaded to set up a Dynamic Services connection to be used throughout the lifetime of the DSAdmin utility. The name depends on the nickname specified in DS_CONNECTION_DESCRIPTOR.

Table 3–1 DSAdmin Utility Configuration Elements (Cont.)

Element	Description
DS_URL	The name of the URL. This URL is used by the specified driver class to open a Dynamic Services connection. The value depends on the nickname specified in DS_CONNECTION_DESCRIPTOR. For each driver class name, there must be a corresponding URL, (for example, `jdbc:oracle:oci8:@db` for a Direct driver class name; `http://host-name:8888/ds/DSServlet` for a Servlet driver class name).
DEFAULT_SERVICE_REQUESTS	The default service requests section. Contains descriptions of the service ID and the default path to the XML request file used for a service in the DSAdmin utility.
DEFAULT_SERVICE_REQUEST	The default service request. Contains the service ID attribute describing the service ID.
DEF_XML	The default path to the XML request file that is used for a specific service corresponding to the given service ID.

In order to use the other drivers, such as HTTP, HTTPS, and JMS, you must complete the advanced installation options (see Chapter 4 for more information). When the Servlet driver is used, requests are sent using HTTP to a Java servlet that directly interacts with a Dynamic Services engine in the same way the Direct driver does. This means that the two drivers may not necessarily share the same execution engine. See Section 1.3.1 through Section 1.3.3 for more information.

Note: Users of the DSAdmin utility should be concerned only about modifying the DEF_XML elements and changing the URL of the predefined driver nicknames so that it points to their database instances, or to the appropriate servlet URL or zones.

Note: The paths specified are relative; thus, you should always execute the DSAdmin utility from the Dynamic Services installation directory.

3.1.3 Running the DSAdmin Utility

Run the DSAdmin utility by executing the command shown in Example 3–2.

Example 3–2 Run the DSAdmin Utility

```
On UNIX systems:
<ORACLE_HOME>/ds/bin/dsadmin

On Windows NT systems:
<ORACLE_HOME>\ds\bin\dsadmin
```

Following each prompt, enter the user name DSSYS, *<DSSYS-password>* (default is DSSYS), and 1 to select the DSConnection nickname named Direct.

The command-line options for running the DSAdmin utility are described in Table 3–2.

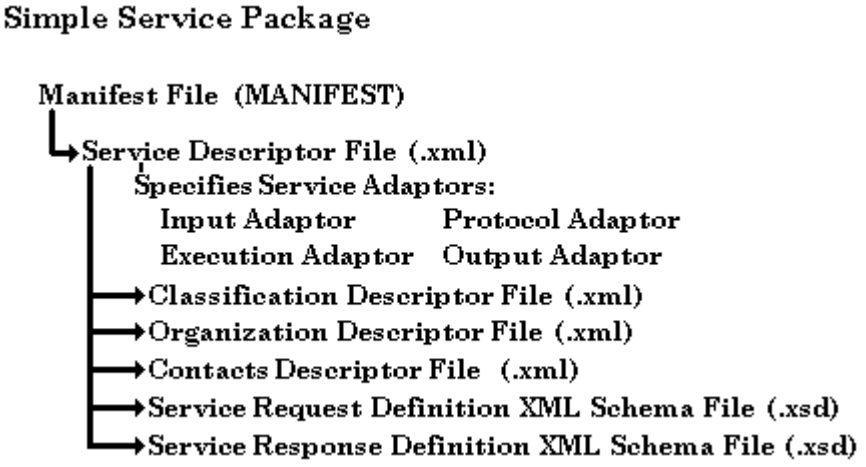
Table 3–2 DSAdmin Utility Command-Line Options

Option	Description
-c <config file>	Allows the DSAdmin utility to load configuration files from any location.
-d	Checks whether all required jars (and appropriate version) are in the CLASSPATH.
-h	Displays help. Lists the DSAdmin utility command-line options.
-i <script file>	Allows actions to be scripted through this option. The DSAdmin utility interprets each command separately and displays the result in standard output.
-s	Executes the DSAdmin utility in silent mode. This option is used only in conjunction with a script file.
-u <username>/<password>@<nickname>	Allows a user name, password, and DSConnection nickname to be specified upon invoking the DSAdmin utility.

3.2 Registering a New Service

A Dynamic Services **simple service package** consists of the group of files shown in Figure 3–1, and is located in a local directory structure on your system.

Figure 3–1 Contents of a Simple Service Package



The MANIFEST file contains a pointer to the service descriptor file. The service descriptor file contains pointers within the appropriate XML tag definitions pointing to the following:

- Classification descriptor file
- Organization descriptor file
- One or more contact descriptor files
- Service request definition file
- Service response definition file

In addition, the service descriptor file specifies the service adaptors to be used (see Chapter 6 for more information about each of these files).

A **compound service package** invokes one or more other services and typically includes one additional file, a jar file, which contains all Java classes and property files needed by the compound service at execution time.

Simple and compound service packages to be used by Dynamic Services must be registered in the **registry**.

Registration is a two-step process:

1. Classify a service under the LDAP category specified in its descriptor.

2. Register the new service package.

The location of this information is in the service. Categories are organized into a Lightweight Directory Access Protocol (LDAP) hierarchical tree, and are therefore defined by a Distinguished Name (DN). Before registering a service package, you must be sure the category that it belongs to exists. If the category does not exist, it must be created.

The entire process of registering the sample service package, YahooPortfolio, is described, starting from category creation (see Section 3.2.1), to registering the service (see Section 3.2.2). You *must* complete instructions described in these two sections, then browse registered services described in Section 3.3, and finally execute a registered service Section 3.4. Successfully completing these sections is the test that your Dynamic Services installation and DSAdmin utility configuration is working properly.

Note: On UNIX systems, you can use the file `<ORACLE_HOME>/ds/demo/services/install_examples.dss` to install a set of sample service packages by entering the following command from your *<ORACLE_HOME>*`/ds` directory:

```
bin/dsadmin -u dssys/<dssys-password>@Direct -i
demo/services/install_examples.dss
```

On Windows NT systems, you can use the file `<ORACLE_HOME>\ds\demo\services\install_examples.dss` to install a set of sample service packages by entering the following command from your *<ORACLE_HOME>*`/ds` directory:

```
bin/dsadmin -u dssys/<dssys-password>@Direct -i
demo\services\install_examples.dss
```

See Section 3.2.1 and Section 3.2.2 for more detailed information on what these scripts do in creating a service package category and registering a service package.

3.2.1 Creating a New Service Package Category

Using a regular text editor, open the service descriptor file of the YahooPortfolio sample service package. The service package is stored in the following directory:

On UNIX systems:
<ORACLE_HOME>/ds/demo/services/YahooPortfolio

```
On Windows NT systems:
<ORACLE_HOME>\ds/demo\services\YahooPortfolio
```

As specified in the MANIFEST file, the location of the service descriptor file is *relative to the service package* in the following file:

```
On UNIX systems:
/www.yahoo.com/dServices/sd/portfolio/yahoo_pfl.xml

On Windows NT systems:
\www.yahoo.com\dServices\sd\portfolio\yahoo_pfl.xml
```

In the service descriptor file header, it is specified that the service category (classification) information is available in an additional XML file stored in the same directory under the file name of `yahoo_pfl_classification.xml`. When viewing this file, note that the following category information is specified for the YahooPortfolio service package:

```
cn=portfolio, cn=finance, cn=business
```

Defined as a DN, this category information must be read in the following way: business is the parent category of finance, which is the parent category of portfolio. To create the needed category in the Dynamic Services engine, start the DSAdmin utility and navigate with the following steps:

1. Start the DSAdmin utility using the following command on UNIX:

   ```
   bin/dsadmin -u dssys/<dssys-password>@Direct
   ```

   Use the following command on Windows NT:

   ```
   bin\dsadmin -u dssys/<dssys-password>@Direct
   ```

   (The Direct driver is the only driver that allows registry manipulation.)

2. Enter **Reg** or **R** to enter the registry subshell (where registry-related operations are performed).
3. Enter **Service** or **S** to enter the service management subshell.

To create the set of categories required by the YahooPortfolio service package, issue the commands shown in Example 3–3.

Example 3–3 Create a Set of Categories Required by the Yahoo Service Package

```
AddCat cn=business
AddCat "cn=finance,cn=business"
```

```
AddCat "cn=portfolio, cn=finance, cn=business"
```

Note: The quotation marks are important in order to treat the entire series of entries as one parameter.

3.2.2 Registering a Service Package

Once the service package categories have been created, you can register a new service package from the same DSAdmin utility menu by issuing the command in the same subshell, as shown in Example 3–4.

Example 3–4 Register a Service Package

```
On UNIX systems:
Register <ORACLE_HOME>/ds/demo/services/YahooPortfolio

On Windows NT systems:
Register <ORACLE_HOME>\ds\demo\services\YahooPortfolio
```

Note: Information presented in Example 3–4 is case-sensitive.

The service package directory is specified as a parameter.

The service package can also be presented in a zip archive file and you would then enter the path to that file instead.

Note: Anytime you make a change to any service-related file, you must reregister that service package using the DSAdmin utility Reregister command.

3.3 Browsing Registered Services

Once a service has been registered, you can browse the list of service IDs in the same registry subshell by entering **Search** (**S**) under the Registry Service menu using the DSAdmin utility. You then need to specify the way in which you want to search the services, by category, by keywords, or by interface. Then, you must specify the matching search pattern. Category-based searches require exact pattern matches because the supplied matching pattern must exist; otherwise, nothing is returned. Example 3–5 shows how to search a list of registered services by category

where the matching pattern includes the service ID of the YahooPortfolio service package.

Example 3–5 Search a List of Registered Services by Category

```
Search CATEGORY "cn=portfolio, cn=finance, cn=business"
```

If a category that contains subcategories is specified, a list of the subcategories is also listed. For example, if the category on which to search is "cn=finance, cn=business", then a subcategory of "cn=portfolio, cn=finance, cn=business" is included in the result list. For example:

```
cn=business
      |
  cn=finance
       |
    cn=portfolio
         |
      urn:com.yahoo:
```

Keyword searches are based on keywords that are supplied in the service descriptor file. Wildcards are allowed. Thus, a keyword search with the pattern "*" returns a list of all the service IDs registered in the Dynamic Services engine as the following steps show.

1. Start the DSAdmin utility using the following command on UNIX:

   ```
   bin/dsadmin -u dssys/<dssys-password>@Direct
   ```

   Use the following command on Windows NT:

   ```
   bin\dsadmin -u dssys/<dssys-password>@Direct
   ```

2. Enter **S** to enter the Search subshell (where search-related operations are performed).
3. Enter **2** to select the keyword classification scheme.
4. Enter the wildcard character ***** (the asterisk) and press Return to begin the search, as Example 3–6 shows.

Example 3–6 Returning a List of Services Using the Wildcard "*" Search

```
        Search for services where KEYWORD="*"...DSREG.search: 2 - *...
DSREG.search: 2 - *... Done

        Search Result:
```

```
            Service IDs:
            ------------
            urn:com.cnnfn:finance.portfolio03
            urn:com.dsFailOver:finance.portfolio03
            urn:com.oanda:conversion.currency03
            urn:com.ual:travel.mileage
            urn:com.yahoo:finance.ipfl04
            urn:com.yahoo:finance.portfolio03

            SubCategories:
            --------------
    Done
```

Finally, searches based on the service interface finds matches with services using the same named request and response schema as those delineated by the interface. The interface name is case-sensitive. For example, if you are searching among all registered services only for those that use the "PortfolioService" interface name, you would enter that search string as shown in Example 3–7.

Example 3–7 Search a List of Registered Services by Interface

```
        Search for services where INTERFACE="PortfolioService"...DSREG.search: 3
 - PortfolioService...
DSREG.search: 3 - PortfolioService... Done

        Search Result:

        Service IDs:
        ------------
        urn:com.cnnfn:finance.portfolio03
        urn:com.dsFailOver:finance.portfolio03
        urn:com.yahoo:finance.portfolio03

        SubCategories:
        --------------
Done
```

3.4 Executing a Registered Service

Once the service has been registered, you can execute it with the following commands:

- Enter **Exit** or **X** twice to return to the top-level shell. If you are starting the shell from the beginning, you can skip this step.
- Enter **Exec** or **E** to enter the Execution shell.
- Enter **Synch** or **S** to perform synchronous execution of a service.

The shell prompts you to choose a service ID from a list that was generated with a keyword search using "*" as the matching pattern. For synchronous execution, the final step is to choose the XML file that contains the request for that service as shown in Example 3–8. The shell waits until the service execution is complete and then, produces the response message.

Note: If you are running within an Intranet, you must set proxy information. If you have already completed the instructions described in Section 3.1.1, then you can execute the `YahooPortfolio` service.

Example 3–8 Execute a Registered Service

```
On UNIX systems:
Exec Synch urn:com.yahoo:finance.portfolio03 <ORACLE_
HOME>/ds/demo/services/YahooPortfolio/pfl_req_ex.xml

On Windows NT systems:
Exec Synch urn:com.yahoo:finance.portfolio03 <ORACLE_
HOME>\ds\demo\services\YahooPortfolio\pfl_req_ex.xml

- Sample Output -
.
.
.
EM.execute service: urn:com.yahoo:finance.portfolio03... Done.
Writing synchronous Response
<PortfolioResp xmlns="http://www.portfolio.org/Portfolio/Response" xmlns:xhtml="
http://www.w3.org/1999/xhtml">
   <Quote>
      <Symbol>ORCL</Symbol>
      <Time>1:53PM</Time>
      <Price>13.61</Price>
      <Change>-11.16%</Change>
      <Volume>48,135,000</Volume>
   </Quote>
   <Quote>
```

```
      <Symbol>AAPL</Symbol>
      <Time>1:53PM</Time>
      <Price>20.82</Price>
      <Change>-3.57%</Change>
      <Volume>4,500,100</Volume>
   </Quote>
</PortfolioResp>

DSAdminShell.Execution>
```

After successfully executing the `YahooPortfolio` service, you can begin developing your own services. See Chapter 6 for information about how to build a service.

To configure and use other deployment views of Dynamic Services, such as HTTP / HTTPS, PL / SQL, or HTTP / Java Messaging Services (JMS), or to set up LDAP as a central master registry, see the appropriate sections in Chapter 4. Chapter 4 also describes other advanced features, such as enabling persistent auditing, manually fine-tuning Dynamic Services properties, and installing the management console.

4

Advanced Installation Options

After the `dsinstall.sql` script has been run, a package named DS_Properties is created as a result of installing the DSSYS schema. Through this package, you can call a setProperty procedure to change system properties of your current Dynamic Services instance. The advanced installation options include the following:

- Enabling PL/SQL interfaces (see Section 4.1)
- Enabling persistent auditing or event monitor services (see Section 4.2)
- Enabling HTTP communication (see Section 4.3)
- Enabling Java Messaging Services (JMS) (see Section 4.4)
- Using LDAP as a central master registry (see Section 4.5)
- Manually fine-tuning Dynamic Services properties (see Section 4.6)

Section 4.1 through Section 4.6 describe these advanced installation options that are provided in the installation package. These options can be invoked with the individual scripts described in each section. Most of these scripts call the DS_Properties.setProperty procedure.

Important: In part of Section 4.3.1, directory paths often show only the UNIX path "/" specification. If you are installing on a Windows NT system, the path specification is "\" and you must make this change, as needed, for the installation to be successful.

4.1 Enabling PL/SQL Interfaces

This installation option coincides with the PL/SQL deployment view described in Section 1.3.2.

1. Run the provided SQL script named `dssys_plsql_init.sql` that is provided to grant the necessary permissions to user DSSYS.

   a. Go to the directory location (`ds/sql` on UNIX systems or `ds\sql` on Windows NT systems) of the `ds_plsql_init.sql` file.

   b. Log in to SQL*Plus as user SYS, as follows:

   ```
   sqlplus SYS/<SYS-password>
   ```

   c. Run the `dssys_plsql_init.sql` script as follows:

   ```
   SQL> @dssys_plsql_init.sql
   ```

   Running this script grants the necessary permissions to user DSSYS.

2. Next, run the SQL script named `ds_plsql.sql` that is provided to install the PL/SQL interface.

   a. Go to the directory location on UNIX systems (`ds/sql`) or on Windows NT systems (`ds\sql`) of the `ds_plsql.sql` file.

   b. Log in to SQL*Plus as user DSSYS as follows:

   ```
   sqlplus DSSYS/<DSSYS-password>
   ```

   c. Run the `ds_plsql.sql` script as follows:

   ```
   SQL> @ds_plsql.sql
   ```

   The following happens upon running this `ds_plsql.sql` script:

   - At the beginning of the script, another script is invoked to load the Dynamic Services library into Oracle JVM, along with its dependent libraries.
   - Next, a subsequent script makes declarations of a PL/SQL package called DynamicServices, mapped to the Java Stored Procedures exposed by the library.
   - The `ds_plsql.log` file is checked to verify the installation of the package.
   - Then, the script is completed.

   Describe the DynamicServices package by issuing the following command at a SQL*Plus prompt.

   ```
   SQL> desc DynamicServices
   ```

A sample, anonymous PL/SQL block is run to test the functions, having already registered the YahooPortfolio service as described in Section 3.2. A sample PL/SQL script `demo/consumer/sample.sql` on UNIX systems or `demo\consumer\sample.sql` on Windows NT systems, found in the Dynamic Services installation directory, tests the DynamicServices package that was just installed.

Refer to Section 5.2 for a more detailed description of how you can use the PL/SQL interface.

4.2 Enabling Persistent Auditing or Event Monitor Services

Dynamic Services offers a persistent auditing feature in which events that can be *thrown* during execution, can be monitored. The monitoring process involves triggering of services to be executed upon receipt of a certain event. These services that get triggered are called **monitor services**. A standalone monitor utility enables the process of auditing these events. Persistent auditing performs among other tasks, service execution logging and event failure notification. See Section 4.2.6 for an example of using the logger monitor service.

4.2.1 Configuring Oracle Advanced Queuing

Because Dynamic Services makes use of Oracle Advanced Queuing for delivering event messages, you must also set the dynamic `init.ora` parameter `aq_tm_processes` for your database instance to a non-zero value (for example, set it to 1) as shown in Example 4–1.

Example 4–1 Set the aq_tm_process init.ora Parameter

```
aq_tm_processes = 1
```

Refer to Oracle Advanced Queuing documentation for more information. Restart the database instance after you modify the `init.ora` file.

4.2.2 Installing Monitor Services

By default, monitor services (which are mostly JDBC services), insert entries into tables under the DSSYS schema. If you changed the password for DSSYS, modify the default DSSYS password in the `MonitorInstall.dss` file to reflect that change.

Before installing the event monitor services, you must first configure the `MonitorInstall.dss` file in the `etc/dsadmin` directory on UNIX systems or `etc\dsadmin` directory on Windows NT systems to point to the database where the monitor services will write information. (Most of these monitor services are database services that just load some processed information into tables.) Make this the same database as the one used for the Dynamic Services engine instance shown in Example 4–2. To make use of the of the notifier service, which is an SMTP service, you must also configure the SMTP mail related properties in the `MonitorInstall.dss` file.

Example 4–2 Configure the MonitorInstall.dss File

```
On UNIX systems:
bin/dsadmin -i etc/dsadmin/MonitorInstall.dss

On Wimdows NT systems:
bin\dsadmin -i etc\dsadmin\MonitorInstall.dss
```

Then, run the `dsmoninstall.sql` script in the `ds/sql` directory on UNIX systems or `ds\sql` directory on Windows NT systems. Running this SQL script will install the tables required by the monitor services.

1. Go to the directory location (`ds/sql` on UNIX systems or `ds\sql` on Windows NT systems) of the `dsmoninstall.sql` file.
2. Log in to SQL*Plus as user DSSYS as follows:

   ```
   sqlplus DSSYS/<DSSYS-password>
   ```

3. Run the `dsmoninstall.sql` script as follows:

   ```
   SQL> @dsmoninstall.sql
   ```

A set of default services is installed from the `etc/services` directory on UNIX systems or `etc\services` directory on Windows NT systems, using the DSAdmin command-line utility with some scripts. These services are invoked by the event monitor utility that is described in Section 4.2.3. It is important to note that none of the monitor services throws events. This prevents an infinite loop from happening where the same monitor services are invoked for the event that they throw.

4.2.3 Using the Event Monitor Utility

In addition to the DSAdmin command-line utility, there is also an event monitor command-line tool called `dsmon` on UNIX systems (`dsmon.bat` on Windows NT

systems). This tool lets you start and stop the event monitor, which executes monitor services upon receipt of events published by the Dynamic Services engine. Monitor services are services that are associated with a monitor and conform to a service interface called EventHandlerTemplate. These services are located in the `etc/services` directory on UNIX systems or `etc\services` directory on Windows NT systems. The correct syntax for running this utility is shown in Example 4–3.

Note: On Windows NT, you must customize the SET ORACLE_HOME line in `dsmon.bat` to point to your <*ORACLE_HOME*>.

Example 4–3 Usage Syntax for Running the Event Monitor Utility

```
dsmon -u dssys/<dssys-password>@Direct -e start
```

Note: Information presented in Example 4–3 is case-sensitive.

Using the event monitor utility, you can connect to a Dynamic Services engine; start or stop the monitor; and have control over the output level of the messages during the execution of the monitor services.

4.2.4 Enabling Persistent Auditing

The next step is to enable persistent auditing. With the default installation in `dsinstall.sql`, event messages are disabled in the properties table. Example 4–4 shows the setProperty PL/SQL procedure calls that enable event logging for the logging and warning event types.

Example 4–4 Connect to the DSSYS Schema as DSSYS User

```
connect dssys/<dssys-password>;
SQL> exec DS_Properties.setProperty('DS_EV_LOGGING_enabled', 'true');
SQL> commit;
```

Configure persistent auditing to enable event messages only for the event types you want.

4.2.5 Starting and Stopping the Event Monitor

Start the event monitor using the command shown in Example 4–5.

Example 4–5 Start the Event Monitor

```
dsmon -u dssys/<dssys-password>@Direct -e start
```

> **Note:** After starting the event monitor on Windows NT systems, the DOS prompt does not display again. You must use another DOS window to issue the stop command shown in Example 4–6.

Stop the event monitor using the command shown in Example 4–6.

Example 4–6 Stop the Event Monitor

```
dsmon -u dssys/<dssys-password>@Direct -e stop
```

When you issue this stop event monitor command, you post a stop request in the queue and the event monitor stops.

4.2.6 Using the Logger Monitor Service (Case Study)

One of the monitor services that is used is called the logger monitor service. It loads a logging event message into a raw log table in the database. The log table is an object table with the object definition as shown in Example 4–7.

Example 4–7 Define the Raw Log Object Table

```
CREATE OR REPLACE TYPE raw_logging_typ AS OBJECT
(
  base           raw_event_typ, -- Raw event type (base)
  operation      VARCHAR2(512), -- Oper: connect, lookup,  execute, session
  status         VARCHAR2(512), -- Status of the operation: open, fail, close
  comm_msg       VARCHAR2(4000) -- Communication Message
);
/
```

The dependent object `raw_event_typ` has a definition as shown in Example 4–8.

Example 4–8 Definition of the Raw Event Object Table

```
CREATE OR REPLACE TYPE raw_event_typ AS OBJECT
(
  time_stamp     DATE,          -- Time stamp of the event
  service_id     VARCHAR2(512), -- Maximum length of a service ID string
  connection_id  VARCHAR2(256), -- Maximum DSConnection ID for a DSE user
```

```
  request_id     VARCHAR2(256), -- Maximum request ID for a DSE user
  consumer_id    VARCHAR2(256), -- Maximum length of a DB user
  engine_id      VARCHAR2(128)  -- Engine identifier (instance of DSE)
);
/
```

With an object table created based on the `raw_logging_typ` object, you can then make SQL queries to give a good view of the logging events that are thrown during service execution, as shown in Example 4–9.

Example 4–9 Make a SQL Query of the Logging Events

```
column timestamp format a14
column service   format a37
column consumer  format a8
column operation format a8
column status    format a6

select TO_CHAR(t.base.time_stamp, 'MM/DD@HH24:MI:SS') as timestamp,
               t.base.consumer_id as consumer,
               t.operation as operation,
               t.base.service_id as service,
               t.status as status
from raw_logging_table t
order by t.base.time_stamp asc;

-- The following is a sample of some logging event information that
-- might display from running the SQL query.
```

```
TIMESTAMP      CONSUMER OPERATIO SERVICE                                  STATUS
-------------- -------- -------- ---------------------------------------- ------
12/07@12:05:20 DSSYS    CONNECT                                           OPEN
12/07@12:05:33 DSSYS    LOOKUP                                            OPEN
12/07@12:05:33 DSSYS    LOOKUP                                            CLOSE
12/07@12:05:36 DSSYS    EXECUTE  urn:com.cnnfn:finance.portfolio03        OPEN
12/07@12:05:53 DSSYS    EXECUTE  urn:com.cnnfn:finance.portfolio03        CLOSE
12/07@12:06:23 DSSYS    CONNECT                                           CLOSE

6 rows selected.
```

There are certain service properties used by the logger monitor service that are set when the logger monitor service is installed. These service properties involve the database URL as well as the schema in the database that contains the raw log tables,

and are therefore necessary for the logging monitor service to function properly. These service properties are described in the script files mentioned in Section 4.2.2.

4.3 Enabling HTTP Communications

This installation option coincides with the HTTP deployment view described in Section 1.3.3.

Dynamic Services can make use of the Apache servlet engine for handling remote HTTP communication between its service consumers and the Dynamic Services engine. To enable HTTP communications, first you must configure the Apache/Jserv servlet engine (see Section 4.3.1), and then configure the DSAdmin utility to use the Dynamic Services HTTP driver, DSHTTPDriver (see Section 4.3.2).

4.3.1 Configuring the Apache/Jserv Servlet Engine

The following instructions assume that you have Apache/JServ installed. Any Web server with a servlet container will work, provided that the changes are done correctly to the correct files. In this step, it is required that you configure your installation of Apache/Jserv to install a new Dynamic Services zone. The following is the list of tasks you must perform (refer to JServ documentation for more information on how to create new zones):

1. Edit the `jserv.conf` file.

   This file is usually found within the `Jserv/etc` directory on UNIX systems or `Jserv\etc` directory on Windows NT systems under your Apache installation.

   Note: For an Oracle9*i* Release 1 (9.0.1) installation, the `jserv.conf` file is located in *<ORACLE_HOME>*`/Apache/Jserv/etc` directory on UNIX and *ORACLE_HOME*`>\Apache\Jserv\etc` on Windows NT.

   Configure a new `ds` mount point by adding the new lines shown as follows:

   ```
   # Oracle Dynamic Services Zone
   ApJServMount /ds /ds
   ```

2. Edit the `jserv.properties` file.

   This file is found in the same directory as the `jserv.conf` file. Make the following modifications:

a. Ensure Jserv is running on Java2.

Modify the wrapper.bin line to show *<JAVA2_HOME>* shown as follows. *<JAVA2_HOME>* is your Java 2 SDK installation directory.

```
wrapper.bin=<JAVA2_HOME>/bin/java
```

b. Create a new ds zone.

Append to the zones line, the ds zone as follows:

```
zones = <existing zones>, ds
```

c. Add a pointer to the ds zone properties as follows:

```
ds.properties=ORACLE_HOME/ds/etc/Apache_JServ/zones/ds/ds.properties
```

d. Update the Jserv classpaths.

Add all the necessary libraries needed by Dynamic Services as shown in the following list of necessary modifications.

```
# ----------------------------------------
# XML / XSD Parser from Oracle
# ----------------------------------------
wrapper.classpath=ORACLE_HOME/lib/xmlparserv2.jar
wrapper.classpath=ORACLE_HOME/lib/xschema.jar
# ----------------------------------------------
# Oracle JDBC Driver (Compliant with JDK 1.2)
# ----------------------------------------------
wrapper.classpath=ORACLE_HOME/jdbc/lib/classes12.zip
# --------------
# JMS / AQ Stuff
# --------------
wrapper.classpath=ORACLE_HOME/rdbms/jlib/jmscommon.jar
wrapper.classpath=ORACLE_HOME/rdbms/jlib/aqapi.jar
# ----------------
# JNDI /LDAP Stuff
# ----------------
wrapper.classpath=ORACLE_HOME/jlib/providerutil.jar
wrapper.classpath=ORACLE_HOME/jlib/ldap.jar
wrapper.classpath=ORACLE_HOME/jlib/jndi.jar
# ----------------------------------------
# JSSE
# ----------------------------------------
wrapper.classpath=ORACLE_HOME/ds/lib/jsse.jar
wrapper.classpath=ORACLE_HOME/ds/lib/jnet.jar
```

```
wrapper.classpath=ORACLE_HOME/ds/lib/jcert.jar
# ---------------
# XMLSQL and XSQL
# ---------------
wrapper.classpath=ORACLE_HOME/rdbms/jlib/xsu12.jar
wrapper.classpath=ORACLE_HOME/lib/oraclexsql.jar
# ------------------------------
# Oracle Dynamic Service Engine
# ------------------------------
wrapper.classpath=ORACLE_HOME/ds/lib/ds.jar
```

e. Set the environment variables.

Ensure that <*ORACLE_HOME*> and <*LD_LIBRARY_PATH*> environment variables are properly set on UNIX and that the <*ORACLE_HOME*> and *<PATH>* environment variables are properly set on Windows NT as follows:

On UNIX, set:

```
wrapper.env=ORACLE_HOME=<your_oracle_home>
wrapper.env=LD_LIBRARY_PATH=<your_oracle_home>/lib
```

On Windows NT, set:

```
wrapper.env=ORACLE_HOME=<your_oracle_home>
wrapper.env=PATH=<your_oracle_home>\bin
```

3. Edit the file *<ORACLE_HOME>*`/ds/etc/Apache_JServ/zones/ds/ds.properties` on UNIX systems or *<ORACLE_HOME>*`\ds\etc\Apache_JServ\zones\ds\ds.properties` on Windows NT systems and make the following modifications:

a. Update the repository location for the Dynamic Services zone.

Change the location of the Dynamic Services jar file as follows:

```
repositories=ORACLE_HOME/ds/lib/ds.jar
```

b. Update Oracle Driver information for DSServlet.

Change the driver to be used by the DSServlet as a servlet property by using the appropriate connection string for your database instance as follows:

```
servlets.default.initArgs=DS_ORCL_URL=jdbc:oracle:thin:@<your-host>:<your-port:<your-SID>
```

4. Restart Apache.

To restart Apache on UNIX, execute the following commands:

```
cd <Apache installation directory>
On UNIX systems:
bin/apachectl restart

On Windows NT systems:
bin\apachectl restart
```

On Windows NT, restart the Apache server from the Start bar. For Oracle9*i* Release 1 (9.0.1), start from your Oracle home, select **Oracle HTTP Server**, then **Start HTTP Server powered by Apache**.

4.3.2 Configuring the DSAdmin Utility to Use the HTTP Driver

After the Apache/JServ installation is completed, you can use the DSHTTPDriver with the DSAdmin utility after you perform the following tasks:

1. Navigate to the `etc/dsadmin/DSAdminConfig.xml` file on UNIX systems or `etc\dsadmin\DSAdminConfig.xml` file on Windows NT systems.
2. Enable the HTTP driver by following the comments for the DS_DRIVERS property. Update the URL used by the DSHTTPDriver by finding the DS_CONNECTION_DESCRIPTOR name = HTTP element and within this element, the element that begins with DS_URL. Change the value to point to the servlet that you just installed.

Note: The URL used by the HTTP driver is an HTTP URL, while the URL used by the Direct driver is a JDBC URL. The rationale is that when the HTTP driver is used, requests are sent using HTTP to the previously described installed Java servlet. This Java servlet directly interacts with a Dynamic Services engine in the same way that the Direct driver does. This means that the two drivers may not necessarily share the same engine.

4.4 Enabling Java Messaging Services (JMS) Communications

This installation option coincides with the JMS deployment view described in Section 1.3.3.

A SQL script named `dsjms_aqinit.sql` is provided to install the JMS option.

1. Go to the directory location (`ds/sql`) on UNIX systems or (`ds\sql`) on Windows NT systems of the `dsjms_aqinit.sql` file.
2. Log in to SQL*Plus as user DSSYS as follows:

   ```
   sqlplus DSSYS/<DSSYS-password>
   ```

3. Run the `dsjms_aqinit.sql` script as follows:

   ```
   SQL> @dsjms_aqinit.sql
   ```

   Running the `dsjms_aqinit.sql` script in a SQL*Plus session as the DSSYS user, creates all the tables and queues necessary for JMS communications.

4.4.1 Configuring and Running the JMS Daemon

To configure and run the JMS daemon, perform the following tasks:

1. Edit the `etc/dsadmin/DSAdminConfig.xml` configuration file on UNIX systems or the `etc\dsadmin\DSAdminConfig.xml` configuration file on Windows NT systems that is used to run the daemon. The following code example shows the specific parameters that you must configure in this configuration file.

```
.
.
.
<DS_CONNECTION_DESCRIPTOR name="JMSDAEMON">
   <annotation>
     -| For Nickname "JMSDAEMON":
      | These are specifications of the JMS driver class
     +| that will be used as well as the URL to be used with it
   </annotation>
   <DS_DRIVER>oracle.ds.driver.DSDirectDriver</DS_DRIVER>
   <DS_URL>jdbc:oracle:thin:@your-host:your-port:your-sid</DS_URL>
   <JMSD_LOG_FILE>logs/jmsd.log</JMSD_LOG_FILE>
   <JMSD_NUM_THREADS>10</JMSD_NUM_THREADS>
</DS_CONNECTION_DESCRIPTOR>
.
.
.
```

> **Note:** There is only a single URL for the database. This is the database that is used to host the request/response queues, as well as the database for the Dynamic Services engine. It must be the same database where the DSSYS schema was installed.

2. Run the following program to start the daemon that listens to all asynchronous requests.

```
On UNIX systems:
bin/dsjmsd -u dssys/dssys@JMSDAEMON -c etc/dsadmin/DSAdminConfig.xml -e start

On Windows NT systems:
bin\dsjmsd -u dssys/dssys@JMSDAEMON -c etc\dsadmin\DSAdminConfig.xml -e start
```

> **Note:** Before running the `dsjmsd.bat` file on Windows NT, check and change the SET ORACLE_HOME line to point to your Oracle home.

Also, in your `DSAdminConfig.xml` file, uncomment the DS_CONNECTION_DESCRIPTOR element for the JMSDAEMON nickname.

For future reference, the following code example shows how to stop the JMS daemon.

```
On UNIX systems:
bin/dsjmsd -u dssys/dssys@JMSDAEMON -c etc/dsadmin/DSAdminConfig.xml -e stop

On Windows NT systems:
bin\dsjmsd -u dssys/dssys@JMSDAEMON -c etc\dsadmin\DSAdminConfig.xml -e stop
```

4.4.2 Configuring the DSAdmin Utility to Enable JMS Communications

Before configuring the DSAdmin utility to enable JMS communications, you must note that for all service consumer applications that want to use the JMS communication path, the database users that represent them must be granted the AQ_Administrator_Role privilege. The client library needs to register itself as an asynchronous subscriber to the response queue for asynchronous executions. Note that DSSYS is already granted that role. To configure the DSAdmin utility to enable JMS communications, perform the following tasks:

1. Navigate to the `etc/dsadmin/DSAdminConfig.xml` configuration file on UNIX systems or the `etc\dsadmin\DSAdminConfig.xml` configuration file on Windows NT systems.
2. Edit the `etc/dsadmin/DSAdminConfig.xml` file on UNIX systems or the `etc\dsadmin\DSAdminConfig.xml` file on Windows systems and update the URL used by the DSJMSDriver by finding the DS_CONNECTION_ DESCRIPTOR name = JMS element, and within this element, the element that begins with DS_URL. Change the value to point to the URL of the database that is hosting the queues. Also, uncomment this JMS nickname DS_ CONNECTION_ DESCRIPTOR element.

During runtime, requests are sent to the request queue in this database. The requests are picked up by the daemon that is communicating with this same database, and used in a service execution that returns a response. That response is submitted to a response queue in the same database, to be picked up asynchronously by the initial request submitter.

4.5 Using Lightweight Directory Access Protocol (LDAP) as a Central Master Registry

As installed in the `dsinstall.sql` script, the instance of the Dynamic Services engine is a standalone instance with its own storage for the registry. To increase scalability, you may want to install multiple Dynamic Services engines communicating with a central master Lightweight Directory Access Protocol (LDAP) registry (see Figure 1–8). First, you must successfully install the Oracle Internet Directory (OID) LDAP server with all the appropriate schemas.

4.5.1 Setting Up LDAP with Oracle Internet Directory

To set up LDAP with OID, you must install OID (see Section 4.5.1.1), and then install the Dynamic Services LDAP schema (see Section 4.5.1.2).

4.5.1.1 Oracle Internet Directory

To install Oracle Internet Directory, run the Oracle Installer of your Oracle9*i* Release 1 (9.0.1) distribution and choose the Oracle9*i* Management and Integration option. Then, select Oracle Internet Directory from the list of displayed products. For more information, refer to Oracle installation instructions.

4.5.1.2 Dynamic Services LDAP Schema

Before proceeding in the installation, verify the following:

- Ensure the `oidmon` instance is running. If not, run the following command to start it.

  ```
  oidmon connect=OIDDB1 sleep=10 start
  ```

  OIDDB1 is the system identifier (SID) of the database instance created by the OID installer.

- Ensure the `oidldapd` server is running. If not, run the following command to start an instance of the OID LDAP server.

  ```
  oidctl connect=OIDDB1 server=oidldapd instance=1 start
  ```

Then, proceed with the installation of the Dynamic Services LDAP schema and issue the following command from a command-line shell:

```
On UNIX systems:
ldapmodify -h oracledev1-sun.us.oracle.com -p 389 -D "cn=orcladmin" -w "welcome"
-v -c -f $<ORACLE_HOME>/ds/ldif/oiddsschema.ldif

On Windows systems:
ldapmodify -h oracledev1-sun.us.oracle.com -p 389 -D "cn=orcladmin" -w "welcome"
-v -c -f $<ORACLE_HOME>\ds\ldif\oiddsschema.ldif
```

Table 4–1 describes the ldapmodify command-line options that can be used for installing the Dynamic Services LDAP schema.

Table 4–1 ldapmodify Command-Line Options for Installing Dynamic Services LDAP Schema

Options	Description
h	Specifies the host machine where OID is running.
p	Specifies the port number to which OID is listening. By default, the port number is 389.
D	Specifies the user name (in Distinguished Name (DN) format defined by LDAP). By default, the admin for OID is "cn=orcladmin".
w	Specifies the password for the user claimed in option "-D". By default, the password for admin is "welcome".
v	Specifies verbose mode.

Table 4–1 ldapmodify Command-Line Options for Installing Dynamic Services LDAP Schema

c	Specifies that all warning or error messages during the installation are delayed from being viewed until the end.
f	Specifies the location of the schema file to be uploaded to OID. In this example, <*ORACLE_HOME*> refers to your Oracle9*i* installation.

The `oiddsschema.ldif` file includes all the necessary steps for the installation of the Dynamic Services schema into OID. These steps are:

1. Create unique attributes used by Oracle Dynamic Services.
2. Create an index on those attributes.
3. Create the object classes.

After successfully installing the Dynamic Services LDAP schema, the next step is to create default entries for Dynamic Services, such as the release number of the product and the root of the User Profile Subtree. Issue the command shown in Example 4–10 to do this.

Example 4–10 Run the ldapmodify Command to Create Default Entries for Dynamic Services

```
On UNIX systems:
ldapmodify -h oracledev1-sun.us.oracle.com -p 389 -D "cn=orcladmin" -w "welcome"
-v -c -f <ORACLE_HOME>/ds/ldif/oiddsdit.ldif

On Windows NT systems:
ldapmodify -h oracledev1-sun.us.oracle.com -p 389 -D "cn=orcladmin" -w "welcome"
-v -c -f <ORACLE_HOME>\ds\ldif\oiddsdit.ldif
```

Note: In this release, the `oiddsdit.ldif` file assumes the DN of OracleContext to be "cn=OracleContext, C=US". Change the DN to one of your choice, if needed.

4.5.2 Configuring Dynamic Services Registry to Use LDAP

In order to change this instance of the Dynamic Services engine into one that communicates with the master LDAP server, you must change some properties in the properties table. This is done by executing the two setProperty PL/SQL procedure calls shown in Example 4–11.

Example 4–11 Configure the Dynamic Services Registry to Use the Master LDAP Server

```
exec DS_Properties.setProperty('oracle.ds.registry.defaultRegistry',
                               'oracle.ds.registry.DSMasterMirrorRegistry');
exec DS_Properties.setProperty('oracle.ds.registry.ldap.providerUrl',
                               'ldap://your.ldap.server:389');
```

The first call instructs the instance to go to a master LDAP server for the central master registry rather than to itself (the default value that was set during installation is 'oracle.ds.registry.DSSimpleRegistry'). The second call points your instance to the correct LDAP server for its registry communications.

You must change `your.ldap.server` to the host name of the machine that is running Oracle Internet Directory.

After you complete the preceding step, perform the following tasks:

1. Run the DSAdmin utility again and go to the `DSAdminShell.Registry.Engine` subshell to register your engine with the central master registry; however, this step is optional and needed only for management purposes.
2. Browse the `DSAdminShell.Registry.Engine` subshell to see the directives available to manage the list of engines that communicate with the central master registry.

4.6 Manual Fine-Tuning of Dynamic Services Properties

Table 4–2 describes the Dynamic Services properties that you can change after installing Dynamic Services.

Table 4–2 Dynamic Services Properties

Property	Description
proxySet	Controls usage of proxy server for HTTP access; (true | false)
proxyHost	Proxy server host name
proxyPort	Proxy server port number
oracle.ds.registry.ldap.providerUrl	URL of the LDAP server to be used as central master registry
oracle.ds.registry.ldap.principal	User name to be used to connect to LDAP server

Table 4–2 Dynamic Services Properties

oracle.ds.registry.ldap.credential	Password to be used to connect to LDAP server
oracle.ds.registry.ldap.rootdn	DN of the root of the Dynamic Services tree in LDAP(cn=OracleDynamicService, cn=Products, <DN of OracleContext>)
cacheSet	Enables or disables service response caching; (true | false)
debugLevel	Controls debug output level; (TERSE | VERBOSE | TRACE)

Note: Both property name and property values are case-sensitive.

The properties are stored in the installed DSSYS schema. To set a property:

1. Connect to the Oracle database as DSSYS using SQL*Plus as follows:

   ```
   sqlplus DSSYS/<DSSYS-password>
   ```

2. Run the setProperty PL/SQL procedure by issuing the following SQL statement:

   ```
   SQL> EXECUTE DS_PROPERTIES.setProperty('<property name>', '<property
   value>');
   ```

3. Display a list of current properties by issuing the following SQL statements:

   ```
   SQL> SET SERVEROUTPUT ON;
   SQL> EXECUTE DS_Properties.show;
   ```

5

Service Consumer Interfaces

This chapter describes how to use the Java and PL/SQL Web application development interfaces.

> **Important:** In Section 5.1.1, directory paths often show only the UNIX path "/" specification. If you are running a Windows NT system, the path specification is "\" and you must make this change, as needed, for configurations to be successful.

5.1 Java Interface for Service Consumers

The client library provides service consumers (application developers) with a Java application programming interface (API) that can be used to access the functions of the Dynamic Services engine. This section illustrates some examples for writing client Java code to create a service request for some of the sample services supplied with Oracle Dynamic Services, and executing them. Before proceeding, make sure the Dynamic Services engine is properly installed, and that you can register and execute services as described in Chapter 3. Also, using the DSAdmin utility, make sure the YahooPortfolio service is registered, because it is used in these examples.

For more information, refer to the supplied sample code in the *<ORACLE_HOME>*`/ds/demo/consumer` directory on UNIX systems or *<ORACLE_HOME>*`\ds\demo\consumer` directory on Windows NT systems and to the supplied Javadoc API (`apidoc.zip` file) in the *<ORACLE_HOME>*`/ds/doc` directory on UNIX systems or in the *<ORACLE_HOME>*`\ds\doc` directory on Windows NT systems.

5.1.1 Setting the Classpath

Make sure your classpath includes all the necessary libraries shown in Example 5–1 (that is, concatenate these paths together with a colon (:) in your classpath, (a semicolon (;) on Windows NT)):

Example 5–1 Include These Dynamic Services Libraries in Your Classpath

```
<ORACLE_HOME>/ds/lib/ds.jar
<ORACLE_HOME>/lib/xmlparserv2.jar
<ORACLE_HOME>/lib/xschema.jar
<ORACLE_HOME>/ds/jlib/providerutil.jar
<ORACLE_HOME>/ds/jlib/ldap.jar
<ORACLE_HOME>/ds/jlib/jndi.jar
<ORACLE_HOME>/rdbms/jlib/xsu12.jar
<ORACLE_HOME>/lib/oraclexsql.jar
<ORACLE_HOME>/ds/lib/jcert.jar
<ORACLE_HOME>/ds/lib/jnet.jar
<ORACLE_HOME>/ds/lib/jsse.jar
<ORACLE_HOME>/jdbc/lib/classes12.zip
<ORACLE_HOME>/rdbms/jlib/jmscommon.jar
<ORACLE_HOME>/rdbms/jlib/aqapi.jar
```

5.1.2 Registering a Service Consumer Application in the Application Profile Registry

Registering an service consumer application in the Dynamic Services application profile registry is a two-step process.

Step 1: Create a new database user in the database instance where the DSSYS schema was installed during the installation process. Example 5–2 shows how a new database user can be created by issuing SQL statements.

Example 5–2 Create a New Database User Using These SQL Statements

```
CONNECT SYSTEM/<system-password>;
CREATE USER serviceconsumer1 IDENTIFIED BY serviceconsumer1;
GRANT CONNECT TO serviceconsumer1;
GRANT DSUSER_ROLE to serviceconsumer1;
```

The third SQL statement lets the service consumer application named `serviceconsumer1` start using Dynamic Services.

Step 2: Using the DSAdmin utility, register the user identity as a new Dynamic Services service consumer application with the following commands:

1. `Enter dsadmin -u DSSYS/<DSSYS-password>@Direct`
2. Enter **Reg** or **R** to enter the registry subshell.
3. Enter **Consumer** or **C** to enter the consumer application profile registry subshell.
4. Enter **Add** or **A** to add a new service consumer application, followed by entering a name of a previously defined database user.

   The following code example shows how to add a service consumer application named `serviceconsumer1` associated with the database user created previously.

   ```
   Add serviceconsumer1
   ```

5. Enter **Grant** or **G** to grant a user privileges to execute services or administer the engine.
6. Enter **Service** or **1** to grant service privileges, followed by the user name to receive the grant, and then select the desired service ID from a list of service IDs. Following the same example, execute the following line to grant the YahooPortfolio service to the service consumer application identified by the name, `serviceconsumer1`.

Example 5–3 Register the Application as a New Dynamic Services Consumer

```
Grant serviceconsumer1 Service urn:com.yahoo:finance.portfolio03
```

In Example 5–3, `urn:com.yahoo:finance.portfolio03` is the service ID of the YahooPortfolio service that is granted to the new user named `serviceconsumer1` that was created in Step 1.

You can try to connect to the Oracle Dynamic Services engine as the new user, `serviceconsumer1`, by executing the command shown in Example 5–4.

Example 5–4 Connect to the Oracle Dynamic Services Engine as User serviceconsumer1

```
dsadmin -u serviceconsumer1/serviceconsumer1@Direct
```

You can display a list of service IDs in the same registry subshell by entering **Search** or **S**. See Section 3.3 for more information.

5.1.3 Opening a Connection to the Dynamic Services Engine

The first step that a service consumer application must perform to work with the Dynamic Services engine is to open a connection to it. This is similar to opening a JDBC connection. There are multiple connection drivers available with Dynamic Services that allow different connection paths from service consumer applications to the engine. Service consumer applications must specify the desired driver, and then operate with the returned connection. The communication protocol used in the driver implementation is completely hidden from service consumer application developers, who will be always writing code using the same API. Some drivers allow asynchronous service requests. Example 5–5 shows how to specify a driver and open a connection for a service consumer application.

Example 5–5 Specify a Driver and Open a Connection for a Service Consumer Application

```
// First open the connection with the Direct driver
DSDriverManager.registerDriver("oracle.ds.driver.DSDirectDriver");
DSConnection dsconn =
DSDriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:ORCL");
dsconn.connect("ServiceConsumer1", "ServiceConsumer1");
```

5.1.3.1 Available Connection Drivers

The following drivers are supplied by Dynamic Services:

- *oracle.ds.driver.DSDirectDriver* for synchronous access to services and service lookup operations
- *oracle.ds.driver.DSHTTPDriver* and *oracle.ds.driver.DSHTTPSDriver* for remote synchronous access to services
- *oracle.ds.driver.DSJMSDriver* for remote synchronous and asynchronous access to services

The following sections describe some important function differences of which service consumer application developers must be aware when using these drivers.

5.1.3.1.1 oracle.ds.driver.DSDirectDriver Use of the Direct driver means the service consumer application assumes the Dynamic Services engine is available in its own classpath, and therefore accessible through direct Java method calls. Through the DSConnection acquired using the Direct driver, service consumer applications can perform service lookup operations as well as synchronous service executions. The URL specified in the getConnection call has to be a valid Oracle JDBC connection string, pointing to the database instance where the DSSYS schema is installed.

5.1.3.1.2 oracle.ds.driver.DSHTTPDriver DSHTTPDriver permits submission of service requests to a remote Dynamic Services engine using HTTP as a communication protocol. DSHTTPDriver assumes the existence of a gateway in the form of a Web server with an installed servlet that can accept service requests. The servlet is installed using the *oracle.ds.comm.protocol.http.DSServlet* class. See Section 4.3 for more information.

5.1.3.1.3 oracle.ds.driver.DSHTTPSDriver DSHTTPSDriver is similar to DSHTTPDriver except that it goes through a secure HTTP (HTTPS) channel when communicating with the remote Dynamic Services engine. In addition, it assumes that the Web server hosting the *oracle.ds.comm.protocol.http.DSServlet* servlet has the HTTPS option enabled.

5.1.3.1.4 oracle.ds.driver.DSJMSDriver DSJMSDriver permits remote synchronous and asynchronous access to services using a Dynamic Services gateway in the form of a JMS daemon. The mode of operation with this driver lets it submit requests asynchronously to an AQ/JMS queue in a remote database. The driver assumes the existence of this JMS daemon and it listens asynchronously to the same queue where requests are submitted. The daemon takes on the role of the Dynamic Services engine and processes the request, generates a response, and submits that response to another queue to which the DSJMSDriver asynchronously listens. On the service consumer application side, therefore, listeners can be registered to be informed when the response is returned.

Note: It is important to note that alternating between the supplied drivers requires no modifications in the service consumer application code other than the registration of the driver itself.

5.1.4 Example: Executing the YahooPortfolio Service

The steps required to execute any service involve:

1. Creating a service request context and the request
2. Making the execution call

Example 5–6 illustrates these steps.

Example 5–6 Request a Service and the Service Execution Call

```
// Create a request with a default request context from the DSConnection.
// Alternatively, the user can create a default request context himself
// and redirect the debugger to somewhere else.
```

```
DSRequest dsReq = dsconn.createDSRequest(myServiceID,
                                         new FileReader(myReqFile));

// Execute synchronously, get the response and print it.
DSResponse dsResp = dsconn.executeSynch(dsReq);
```

In Example 5–6, the service request is read from a file. Any `java.io.Reader` can be used to supply the XML request document.

Example 5–7 describes the example request of the `YahooPortfolio` service in the `pfl_req_ex.xml` file in the `ds/demo/services/YahooPortfolio` directory on UNIX systems or in the `ds\demo\services\YahooPortfolio` directory on Windows NT systems.

Example 5–7 Request the YahooPortfolio Service

```
<?xml version="1.0"?>
<!-- Example request of the YahooPortfolio service -->
<PortfolioReq  xmlns="http://www.portfolio.org/Portfolio/Request"
         xmlns:xsi = "http://www.w3.org/1999/XMLSchema/instance"
         xsi:schemaLocation = "http://www.portfolio.org/Portfolio/Request
                               http://www.portfolio.org/Portfolio/Request pfl_req.xsd">
  <Symbol>ORCL</Symbol>
  <Symbol>AAPL</Symbol>
</PortfolioReq>
```

The supplied XML request document has to comply with request syntax defined for the YahooPortfolio service.

5.1.5 Displaying Service Response

Once a service response has been obtained, its content can be parsed by the Oracle XML parser and printed as shown in Example 5–8.

Example 5–8 Display a Service Response

```
StringWriter sw = new StringWriter();
dsResp.writeResponse(sw);

DOMParser xmlp = new DOMParser();
xmlp.parse( new StringReader(sw.toString()));

XMLDocument xmldoc = xmlp.getDocument();
xmldoc.print( new PrintWriter(System.err));
```

5.1.6 Service Consumer Application Sessions

Within the life cycle of a Dynamic Services connection, service consumer applications can execute multiple services. Each of these services can actually create a session with the remote service provider. For example, a service connecting to a Web site can receive as part of the response an HTTP cookie that has to be supplied with every request that follows.

Before executing a set of services, Dynamic Services allows service consumer applications to create a session and execute a set of services within the session so that all the session context (for example, HTTP cookies or database connections) are preserved for that session only. By calling the DSConnection.openSession() method, service consumer applications obtain an *opaque* session identifier. To continue the session, service consumer applications must set the session identifier in the header of those service requests that are to be executed within the session. Corresponding DSResponses contain header information about the session to which they belong. To close a session, service consumer applications can use the DSConnection.closeSession() method, which releases all the resources related to the specified session. Refer to the sample Java code for details.

The information stored for the session (for example, HTTP cookies and database connections) is not persistent across startup and shutdown of the Dynamic Services engine. This information is stored in memory and it persists only through the life cycle of the host JVM where the Dynamic Services engine is running.

It is the responsibility of the service consumer (application) to close any session that it created so the associated resources are released. Closing a Dynamic Services connection does not close the service consumer sessions and release their resources.

5.2 PL/SQL Interface for Service Consumers

The PL/SQL DynamicServices package defines the PL/SQL interface for service execution. The PL/SQL DynamicServices package is defined with *invoker's* privileges; therefore, to access it in a PL/SQL block that is defined with *definer's* privileges, the package and related types must be explicitly granted to service consumers as shown in Example 5–9.

Example 5–9 Use These Grant Statements to Access the PL/SQL DynamicServices Package

```
GRANT EXECUTE ON DSSYS.DYNAMICSERVICES TO serviceconsumer1;
GRANT EXECUTE ON DSSYS.XML_ELEM_NAMES TO serviceconsumer1;
GRANT EXECUTE ON DSSYS.XML_ELEM_VALS TO serviceconsumer1;
```

To easily create a service consumer application that uses Dynamic Services, you can inspect the `createPLSQLConsumer.sql` file in the `demo/consumer` directory. For more details about users and the database, refer to Oracle9*i* documentation.

As described in Section 1.3.2, in a PL/SQL deployment of Dynamic Services, the Dynamic Services engine runs in the Oracle9*i* JVM, and its functions are exposed as a set of Java stored procedures through a PL/SQL interface (see Example 5–10). A service consumer application makes use of the services through PL/SQL calls to these procedures and functions shown in Example 5–10.

Example 5–10 PL/SQL Interface for Dynamic Services

```
-- This procedure initializes the Dynamic Services engine within JServer
-- and opens a Dynamic Services connection. It is a prerequisite before any
-- kind of execution is done.
PROCEDURE open;

-- This closes the Dynamic Services connection opened by the open function.
-- If no connection is opened, this will throw a TearDownException error.
PROCEDURE close;

-- This function executes a service with a given service identifier and
-- a request in the form of an XML document.
-- It synchronously executes the service and
-- returns the response in the form of an XML document as a VARCHAR2 type.
FUNCTION execute(service_id VARCHAR2, request VARCHAR2) RETURN VARCHAR2;

-- This executes a service with a given service identifier and
-- two CLOB locators. It reads in the request CLOB and starts
-- a synchronous execution. Upon finishing, it writes the result
-- into the response CLOB locator that is passed in.
PROCEDURE execute(service_id VARCHAR2, request CLOB, response CLOB);

-- Utility method: The supplied string has to be an XML element.
-- It will take the XML document and traverse down an entry
-- in the supplied arrays for each element in the document.
-- In the keys array, it will store the path of the element where the slash
-- (/) is used to separate the child files. The corresponding entry
-- in the VALS array will have the value of the element.
PROCEDURE flatXML(szXML VARCHAR2,
                  keys IN OUT DSSYS.XML_ELEM_NAMES,
                  vals IN OUT DSSYS.XML_ELEM_VALS);

-- Utility method to handle the array returned by flat XML.
```

```
  -- It will take the supplied key, iterate over the
  -- keys arrays, and if it finds a match, return the
  -- corresponding value from the VALS array.
  FUNCTION getXMLValue(key VARCHAR2,
                       keys IN DSSYS.XML_ELEM_NAMES,
                       vals IN DSSYS.XML_ELEM_VALS)
     RETURN VARCHAR2;
```

Example 5–11 shows some PL/SQL sample code from the `sample.sql` file in the `demo/consumer` directory that illustrates a typical scenario where the PL/SQL DynamicServices package can be used.

Example 5–11 Sample Code to Use the Dynamic Services PL/SQL Interface Package

```
-- Some output specifications
SET SERVEROUTPUT ON SIZE 20000;
CALL DBMS_JAVA.SET_OUTPUT(20000);

-- Anonymous block
DECLARE

   -- Service Execution
   ds_req   VARCHAR2(512);  -- A request in the form of an XML document.
   ds_resp  VARCHAR2(4000); -- A response in the form of an XML document.
   ds_svcid VARCHAR2(128);  -- A string that tells which service to execute.

   -- For response processing
   ds_elem_names DSSYS.XML_ELEM_NAMES; -- Element Names VARRAY
   ds_elem_vals  DSSYS.XML_ELEM_VALS;  -- Element Values VARRAY

BEGIN

   -- First connect; must do this before any execution
   DSSYS.DynamicServices.open();

   -- Set up the service ID
   ds_svcid := 'urn:com.yahoo:finance.portfolio03';

   -- Set up the service request
   ds_req :=
     '<PortfolioReq xmlns="http://www.portfolio.org/Portfolio/Request">'||
     '  <Symbol>ORCL</Symbol>'||
     '</PortfolioReq>';

   -- Execute the service
```

```
    ds_resp := DSSYS.DynamicServices.execute(ds_svcid, ds_req);

    -- Close connection
    DSSYS.DynamicServices.close();

    -- Print the response (Banner)
    DBMS_OUTPUT.PUT_LINE('-------------------------');
    DBMS_OUTPUT.PUT_LINE('Dynamic Services Response');
    DBMS_OUTPUT.PUT_LINE('-------------------------');

    -- First flatten out the XML
    DSSYS.DynamicServices.flatXML(ds_resp, ds_elem_names, ds_elem_vals);

    -- Which symbol did we try to check?
    DBMS_OUTPUT.PUT_LINE('Value of "/PortfolioResp/Quote/Symbol" is '||
      DSSYS.DynamicServices.getXMLValue('/PortfolioResp/Quote/Symbol',
      ds_elem_names, ds_elem_vals));

    -- What's its price?
    DBMS_OUTPUT.PUT_LINE('Value of "/PortfolioResp/Quote/Price" is '||
      DSSYS.DynamicServices.getXMLValue('/PortfolioResp/Quote/Price',
      ds_elem_names, ds_elem_vals));

END;
/
```

The connected database user is the service consumer application connecting to the service engine. Refer to Section 5.1.2 on how to register a service consumer application with the Dynamic Services application profile registry.

For a more extensive sample that makes use of the currency service, refer to the `demo/consumer/currency.sql` file on UNIX systems or to the `demo\consumer\currency.sql` file on Windows NT systems.

6

Service Development Guide

In this chapter, the process of service development is described as well as how you can test a service after you build one.

A **service** is a component within the Internet computing model that delivers a specialized value-added function. A service is bundled into a **simple service package** (see Figure 3–1) and structured as a local directory containing at least:

- A `MANIFEST` file that points to the service descriptor XML file
- A service descriptor XML file that is the key XML document that describes the service and points to the following descriptor .xml files and .xsd files within its service header section:
  - One classification descriptor .xml file containing suggested classification information from the service provider
  - One organization descriptor .xml file containing company information about the service provider
  - One or more contact descriptor .xml files containing contact information from the service provider
  - One request definition (.xsd) file for the service interface specification
  - One response definition (.xsd) file for the service interface specification

The service descriptor file also describes in its service body section how the four types of service adaptors are to be used to do any of the following:

- Handle the submitted service request (input adaptor).
- Adapt the XML service request to the communication protocol used by the remote service provider (protocol adaptor).
- Determine execution flow (if desired) of a service (execution adaptor).

- Transform the raw response returned by the remote service provider into a service XML response (output adaptor).

A **compound service package** invokes one or more other services and has everything a simple service package has plus it typically includes a *jar* file containing all Java classes and property files needed by the compound service.

The `MANIFEST` file is expected to be found in the root directory of the simple or compound service package and with the name `MANIFEST` (uppercase, case-sensitive on Solaris systems; initial capitalization, non case-sensitive on Windows NT systems). The `MANIFEST` file is a text file where the first non-empty line should specify a URL link to the service descriptor XML file. If a link starts with a slash (/), it indicates the link is an absolute link with respect to the root directory of the current service package. The root directory is interpreted to be the root of the directory structure for the service package.

6.1 Quick Start

You can quickly start developing your own service by following the steps described in this section. These steps are necessary to build a simple HTTP service. Later, you can enhance your service after reading about some more advanced concepts in other sections of this chapter. The service that you will build is a simple HTTP service that gets stock quotes from Yahoo.com.

The tasks to complete this quick-start service development tutorial are as follows:

1. Create a service package (see Section 6.1.1).
2. Edit the service provider organization and contact XML files (see Section 6.1.2).
3. Edit the service provider classification XML file (see Section 6.1.3).
4. Create your XML schema file for the service request definition (see Section 6.1.4).
5. Create your XML schema file for the service response definition (see Section 6.1.5).
6. Edit the service descriptor file, including both the service header and the service body sections (see Section 6.1.6).
7. Test the execution of your service (see Section 6.1.7).

Note: The *<ORACLE_HOME>*/ds/etc/xsd directory on UNIX systems or the *<ORACLE_HOME>*\ds\etc\xsd directory on Windows NT systems contains the XML schema files for the service descriptor and supplied adaptors. Refer to the files in this directory for more information.

6.1.1 Creating a Service Package

Perform the following steps to create your service package:

1. Copy the entire demo/services/SampleService directory on UNIX systems or the demo\services\SampleService directory on Windows NT systems into a new directory using your name, for example, demo/services/myService on UNIX systems or demo\services\myService on Windows NT systems.

   This step creates a default service package. You can modify the name of the subdirectories to reflect the nature of the service you want to build. In this tutorial, you will make the following changes shown in Example 6–1.

For UNIX Systems:

Example 6–1 Create a Default Service Package

```
cp -r demo/services/SampleService demo/services/myService
cd demo/services/myService
mv SampleProvider www.yahoo.com

cd www.yahoo.com/dServices
mv SampleService portfolio
mv SampleOrg.xml YahooOrg.xml
mv SampleContact.xml YahooContact.xml

cd portfolio
mv SampleService.xml ypfl.xml
mv SampleServiceClassification.xml ypflClass.xml
```

For NT Systems:

a. Copy the entire demo\services\SampleService directory into a new directory using your name, for example, demo\services\myService.

b. Navigate to the `demo\services\myService` directory and rename `SampleProvider` to `www.yahoo.com`.

c. Navigate to the `www.yahoo.com\dServices` directory and rename `SampleService` to `portfolio`, `SampleOrg.xml` to `YahooOrg.xml`, and `SampleContact.xml` to `YahooContact.xml`.

d. Navigate to the `portfolio` subdirectory and rename `SampleService.xml` to `ypfl.xml` and `SampleServiceClassification.xml` to `ypflClass.xml`.

2. Update the `MANIFEST` file in the `demo/services/myServices` directory on UNIX systems or the `demo\services\myServices` directory on Windows NT systems to contain the line shown in Example 6–2.

Example 6–2 Update the MANIFEST File

```
On UNIX systems:
/www.yahoo.com/dServices/portfolio/ypfl.xml

On Windows NT systems:
\www.yahoo.com\dServices\portfolio\ypfl.xml
```

Step 2 lets the service package point to the correct service descriptor file that you will edit soon. Notice that all paths used in this quick-start document are relative to the `demo/services/myService` directory on UNIX systems or the `demo\services\myService` directory on Windows NT systems.

6.1.2 Service Provider -- Organization and Contacts XML Files

Recall that the `YahooOrg.xml` file and the `YahooContact.xml` file described in Section 6.1.1, reside in the directory `/www.yahoo.com/dServices` on UNIX systems or `\www.yahoo.com\dServices` on Windows NT systems. These files contain service provider information about the organization and contacts for this particular service.

- Edit the `YahooOrg.xml` file to appear as shown in Example 6–3.

Example 6–3 Edit the YahooOrg.xml File

```
<?xml version="1.0"?>
<!-- Fully scope information for good practice -->
<dsOrg:ORGANIZATION
  xmlns:dsOrg="http://www.oracle.com/ds/2000/SERVICE_DESCRIPTOR/ORGANIZATION">
  <dsOrg:NAME>Yahoo!</dsOrg:NAME>
```

```
  <dsOrg:COPYRIGHT>(c) Yahoo!, 2000</dsOrg:COPYRIGHT>
  <dsOrg:URL>http://www.yahoo.com</dsOrg:URL>
  <dsOrg:LOGOURL>http://us.yimg.com/i/fi/main4.gif</dsOrg:LOGOURL>
</dsOrg:ORGANIZATION>
```

- Edit the `YahooContact.xml` file to appear as shown in Example 6–4.

Example 6–4 Edit the YahooContact.xml File

```
<?xml version="1.0"?>
<!-- Fully scope information for good practice -->
<dsCt:CONTACT
  xmlns:dsCt="http://www.oracle.com/ds/2000/SERVICE_DESCRIPTOR/CONTACT">
  <dsCt:NAME>bar1</dsCt:NAME>
  <dsCt:EMAIL>bar1@yahoo.com</dsCt:EMAIL>
  <dsCt:PHONE>(000)000-0000</dsCt:PHONE>
  <dsCt:FAX>(000)000-0000</dsCt:FAX>
  <dsCt:PAGER>(000)000-0000</dsCt:PAGER>
  <dsCt:MOBILE>(000)000-0000</dsCt:MOBILE>
</dsCt:CONTACT>
```

6.1.3 Service Classification XML File

The `ypflClass.xml` file described in Section 6.1.1 resides in the directory `/www.yahoo.com/dServices/portfolio/` on UNIX systems or `\www.yahoo.com\dServices\portfolio\` on Windows NT systems. This file should contain classification information of your service.

Edit the `ypflClass.xml` file to appear as shown in Example 6–5.

Example 6–5 Edit the ypflClass.xml File

```
<?xml version="1.0"?>
<!-- Fully scope information for good practice -->
<dsCls:CLASSIFICATION
  xmlns:dsCls="http://www.oracle.com/ds/2000/SERVICE_DESCRIPTOR/CLASSIFICATION">
   <dsCls:CATEGORY>cn=portfolio, cn=finance, cn=business</dsCls:CATEGORY>
   <dsCls:KEYWORDS>portfolio,stocks,finance</dsCls:KEYWORDS>
</dsCls:CLASSIFICATION>
```

Note: The category section follows the Lightweight Directory Access Protocol (LDAP) Distinguished Name (DN) (backwards tree) convention. The category specified must exist in the registry before you can register the service.

6.1.4 Service Interface Specification -- Request Definition

Before editing the service descriptor, you must understand how requests are defined.

- Start by looking at a typical HTML form. Example 6–6 shows a portion of an HTML form that you can find on the Yahoo Web site accessed from `http://quote.yahoo.com`.

Example 6–6 Examine a Typical HTML Form

```
<form method=get action="/q"><nobr>
  <input type=text size=25 name=SymbolList>
  <input type=submit value="Get Quotes"> 
  ...
</form>
```

The form takes one input called SymbolList and an HTTP GET request is made to `http://quote.yahoo.com/q` when you click **Submit**. An HTML page returns the stock quotes of the symbols that are specified in the input called SymbolList.

- Make the HTTP form into a service in the Dynamic Services framework.

  The form takes one input called SymbolList. From there, you can expect the service consumer application to pass in only one argument, and generate an XML schema file for your request, such as shown in Example 6–7.

Example 6–7 Generate an XML Schema File for the Service Request

```
<?xml version ="1.0"?>
<!-- Input schema of the currency service -->
<schema xmlns = "http://www.w3.org/1999/XMLSchema"
        targetNamespace = "http://www.portfolio.org/Portfolio/Request"
        xmlns:pflReq = "http://www.portfolio.org/Portfolio/Request">

  <element name = "PortfolioReq">
    <complexType content = "elementOnly">
      <sequence>
```

```
        <!-- Use a more user-friendly name as a symbol and an augmented
             string type called Ticker to restrict its format. Have a
             default as well because XML Schema allows for it. Also,
             restrict it so there are 1 or more symbols at least. -->
        <element
          name = "Symbol" type="pflReq:Ticker"
          default="ORCL" minOccurs="1" maxOccurs="*"/>
      </sequence>
    </complexType>
  </element>

  <simpleType name = "Ticker" base = "string">
    <pattern value="[^\s]+" />
  </simpleType>
</schema>
```

In the use of the service, an XML request must conform to this schema to be used correctly.

- Create your request XML file and place it in the directory `/www.yahoo.com/dServices/portfolio/` on UNIX systems or `\www.yahoo.com\dServices\portfolio\` on Windows NT systems and name it `ypfl_req.xsd`.

6.1.5 Service Interface Specification -- Response Definition

When the HTTP GET request is made, the HTML page shown contains the actual stock quote that you want.

- Examine the code that contains the price for the stock symbol ORCL in Example 6–8.

Example 6–8 Examine the Code and Note the Stock Symbol ORCL

```
<tr align=right>
  <!-- "Symbol" -->
  <td nowrap align=left><a href="/q?s=ORCL&d=t">ORCL</a></td>
  <!-- "Time" -->
  <td nowrap align=center>12:14PM</td>
  <!-- "Price" -->
  <td nowrap><b>82 <sup>15</sup>/<sub>16</sub></b></td>
  <!-- "Change" -->
  <td nowrap>+1 <sup>3</sup>/<sub>4</sub></td>
  <td nowrap>+2.16%</td>
  <!-- "Volume" -->
```

```
<td nowrap>6,218,900</td>
<td nowrap align=center><small>
```

- Transform the HTML into an XML document that a service consumer application can use. Determine what useful information should be extracted.
- Create another XML schema file, this time for the service response, as shown in Example 6–9.

Example 6–9 Create an XML Schema File for the Service Response

```
<?xml version ="1.0"?>
<!-- Input schema of the currency service -->
<schema xmlns = "http://www.w3.org/1999/XMLSchema"
        targetNamespace = "http://www.portfolio.org/Portfolio/Response"
        xmlns:pflResp = "http://www.portfolio.org/Portfolio/Response">
  <!-- This is the input value in which the input should be specified. -->
  <element name = "PortfolioResp">
    <complexType content = "elementOnly">
      <element name = "Quote" minOccurs="1" maxOccurs="*">
        <complexType content = "elementOnly">
          <sequence>
            <element name = "Symbol" type="pflResp:Ticker" />
            <element name = "Time" type="string" />
            <element name = "Price" type="string" />
            <element name = "Change" type="string" />
            <element name = "Volume" type="string" />
          </sequence>
        </complexType>
      </element>
    </complexType>
  </element>

  <simpleType name = "Ticker" base = "string" >
    <pattern value="[^\s]+" />
  </simpleType>
</schema>
```

You have decided that the symbol, time, price, change of last trade, and the volume are all useful pieces of information that you can gather from the HTML page. Consequently, you model your response using the previous schema.

- Create your response XML file and place it in the `/www.yahoo.com/dServices/portfolio/` directory on UNIX systems or

\www.yahoo.com\dServices\portfolio\ directory on Windows NT systems and name it ypfl_resp.xsd.

6.1.6 Editing the Service Descriptor

Next, the steps to modify the service descriptor ypfl.xml file in the directory /www.yahoo.com/dServices/portfolio/ on UNIX systems or \www.yahoo.com\dServices\portfolio\ on Windows NT systems are described. The beginning of the service descriptor, with namespaces sd for all service descriptor tags and xlink for all your document links that use XLink attributes, is shown in Example 6–10.

Example 6–10 Examine the Beginning of the Service Descriptor

```
<sd:SERVICE_DESCRIPTOR
  xmlns:sd="http://www.oracle.com/ds/2000/SERVICE_DESCRIPTOR"
  xmlns:xlink="http://www.w3.org/1999/xlink">
```

Note: Oracle Corporation recommends that you fully qualify the elements in the service descriptor document using the sd prefix and referring to the following namespace:

http://www.oracle.com/ds/2000/SERVICE_DESCRIPTOR

6.1.6.1 Service Header

Modify your service header as shown in Example 6–11, and read the comments that tell you what must be changed when you build the YahooPortfolio service.

Example 6–11 Modify the Service Header

```
<sd:SERVICE_HEADER>
<!-- In the NAMING section, the only thing you really need to modify is
     the ID field. It has to uniquely identify your service and must be a
     universal resource name (URN). But modify the rest as you see fit. -->
<sd:NAMING>
  <sd:ID>urn:com.yahoo:finance.portfolio03</sd:ID>
  <sd:NAME>Yahoo Portfolio service</sd:NAME>
  <sd:DESCRIPTION>Find current prices for stocks</sd:DESCRIPTION>
</sd:NAMING>

<sd:PACKAGE>
  <sd:VERSION>1.0</sd:VERSION>
  <sd:RELEASEDATE>05-MAY-2000</sd:RELEASEDATE>
```

```
  <sd:UPDATEURL>http://www.yahoo.com/dServices/pfl.zip</sd:UPDATEURL>
</sd:PACKAGE>
<sd:DEPLOYMENT>
  <!-- Point the classification file to the file that you edited previously in
       Section 6.1.3 and note that the path starts from the directory of the
       service package. -->
  <sd:CLASSIFICATION
    xlink:href="/www.yahoo.com/dServices/portfolio/ypflClass.xml"/>

  <!-- Also, change the caching parameters to set cache expiration in
       seconds, or to specify that the cache has session knowledge. -->
  <sd:CACHING>
    <!-- Expiration in seconds. -->
    <sd:MAX_AGE>60</sd:MAX_AGE>
    <!-- Will the cache be session-aware? -->
    <sd:SESSION_PRIVATE>true</sd:SESSION_PRIVATE>
    <!-- This Boolean field tells the engine to allow the expiration of
         the cache to be controlled by the underlying protocol. Specifying
         a value of true would make the engine ignore the MAX_AGE tag. -->
    <sd:USE_PROTOCOL>false</sd:USE_PROTOCOL>
  </sd:CACHING>

</sd:DEPLOYMENT>
<sd:PROVIDER>
  <!-- This is mandatory and should point to the organization file
       that you edited previously in Section 6.1.2. -->
  <sd:ORGANIZATION xlink:href="/www.yahoo.com/dServices/YahooOrg.xml"/>

  <!-- This is mandatory (at least one contact element in the contacts
       section), and should point to the contact file that you
       edited previously in Section 6.1.2. -->
  <sd:CONTACTS>
    <sd:CONTACT xlink:href="/www.yahoo.com/dServices/YahooContact.xml"/>
  </sd:CONTACTS>
</sd:PROVIDER>
<sd:INTERFACE>
  <!-- Change this to your own service interface (made up of a request/
       response schema specification pair). We will not put Yahoo here
       because maybe other organizations can have the same kind of service,
       which can be used in a failover scenario. -->
  <sd:NAME>PortfolioService</sd:NAME>
  <!-- This is mandatory; point this to the XML schema file that you
       created previously in Section 6.1.4. -->
  <sd:INPUT_SCHEMA
    xlink:href="/www.yahoo.com/dServices/portfolio/pfl_req.xsd"/>
```

```
  <!-- This is mandatory; point this to the XML schema file that you
       created previously in Section 6.1.5. -->
  <sd:OUTPUT_SCHEMA
   xlink:href="/www.yahoo.com/dServices/portfolio/pfl_resp.xsd"/>
</sd:INTERFACE>
</sd:SERVICE_HEADER>
```

6.1.6.2 Service Body

This section describes the service body from the same YahooPortfolio service. The fields that you must change to modify your own service are described in this section, starting from the service body as shown in Example 6–12.

Example 6–12 Look for the Beginning of the Service Body

```
<sd:SERVICE_BODY>
```

6.1.6.2.1 Input Handling and Input Adaptor Specification

This section describes the input section of the service body.

Modify the input section of the service body of your descriptor to appear as shown in Example 6–13.

Example 6–13 Modify the Input Section of the Service Body

```
    <sd:INPUT>
  <!-- Aliases are what map the XML requests that the service consumer
       will supply when using the service, to the parameters on
       the HTML form of our Web service. -->
  <sd:ALIASES>
    <sd:ALIAS>
      <!-- This name is just a variable name; all references to it in
           the service descriptor will access the same value. -->
      <sd:NAME>SymbolList</sd:NAME>
      <!-- No namespace prefix is needed, as the request transformed by
           inputadaptor has no namespace. -->
      <sd:VALUE>{@xpath:value=/PortfolioReq/SymbolList}</sd:VALUE>
    </sd:ALIAS>
  </sd:ALIASES>
  <sd:ADAPTOR>
    <sd:NAME>oracle.ds.engine.ioa.DSXSLTInputAdaptor</sd:NAME>
    <sd:PARAMETERS>
      <xiaParams:XSLT_IA_PARAMS
        xmlns:xiaParams="http://www.oracle.com/ds/2000/XSLT_IA_PARAMS">
        <xiaParams:XSLT>
```

```
            <xsl:stylesheet
                 version="1.0"
                 xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
                 xmlns:xhtml="http://www.w3.org/1999/xhtml"
                 xmlns:pflreq="http://www.portfolio.org/Portfolio/Request">
              <xsl:template match="/">
                <pflreq:PortfolioReq>
                  <xsl:apply-templates select="pflreq:PortfolioReq"/>
                </pflreq:PortfolioReq>
              </xsl:template>
              <xsl:template match="pflreq:PortfolioReq">
                <pflreq:SymbolList>
                  <xsl:for-each select="pflreq:Symbol">
                    <xsl:value-of select="concat(text(), ',')"/>
                  </xsl:for-each>
                </pflreq:SymbolList>
              </xsl:template>
            </xsl:stylesheet>
          </xiaParams:XSLT>
        </xiaParams:XSLT_IA_PARAMS>
      </sd:PARAMETERS>
    </sd:ADAPTOR>
</sd:INPUT>
```

6.1.6.2.2 Protocol Adaptor Specifications The protocol adaptor specifications contain information on how to map the aliases defined in Example 6–13 with the actual HTTP GET request.

Modify the protocol section of the service body of your service descriptor as shown in Example 6–14.

Example 6–14 Modify the Protocol Section of the Service Body

```
<sd:PROTOCOL>
  <sd:ADAPTOR>
    <sd:NAME>oracle.ds.engine.pa.http.DSHTTPProtocolAdaptor</sd:NAME>
    <sd:DRIVER>java.net.URLConnection</sd:DRIVER>
    <sd:PARAMETERS>
      <hpParams:HTTP_PA_PARAMS
        xmlns:hpParams="http://www.oracle.com/ds/2000/HTTP_PA_PARAMS">
        <hpParams:Method>GET</hpParams:Method>
        <hpParams:URL>quote.yahoo.com/query</hpParams:URL>
        <hpParams:QueryStringParameters>
        <hpParams:QueryStringParameter
          name="SymbolList">{@SymbolList}</hpParams:QueryStringParameter>
```

```
        </hpParams:QueryStringParameters>
      </hpParams:HTTP_PA_PARAMS>
    </sd:PARAMETERS>
  </sd:ADAPTOR>
</sd:PROTOCOL>
```

In the QueryStringParameters and QueryStringParameter sections, the HTTP GET parameter SymbolList is mapped to your alias (which is conveniently also called SymbolList). For more detailed descriptions of the protocol adaptor section, see Section 6.3.2.

6.1.6.2.3 Execution Adaptor Specifications There are no special execution adaptors that you will use in this service, so none is specified. For more detailed descriptions of the execution adaptor section, see Section 6.3.2.

6.1.6.2.4 Output Adaptor Specifications The output adaptor specifications contain information on how the raw output from the Web service (HTML) is to be transformed into the structured XML format that is described with your response XML Schema file described previously.

Modify your service descriptor to contain what is shown in Example 6–15.

> **Note:** If the raw response is an XML document and if no stylesheet is supplied, the service will return the raw response without any processing.

Example 6–15 Modify the Output Section of the Service Body

```
<sd:OUTPUT>
  <sd:ADAPTOR>
    <sd:NAME>oracle.ds.engine.ioa.DSXSLTOutputAdaptor</sd:NAME>
    <sd:PARAMETERS>
      <xoParams:XSLT_OA_PARAMS
        xmlns:xoParams="http://www.oracle.com/ds/2000/XSLT_OA_PARAMS">
        <xoParams:XSLT>
          <xsl:stylesheet version="1.0"
               xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
               xmlns:pflResp="http://www.portfolio.org/Portfolio/Response"
               xmlns:xhtml="http://www.w3.org/1999/xhtml">

            <xsl:template match="/">
              <pflResp:PortfolioResp>
                <xsl:apply-templates
```

```
                    xmlns="http://www.w3.org/1999/xhtml"
                    select="html/body/center/table[5]/tr[1]/td/table"/>
                </pflResp:PortfolioResp>
              </xsl:template>
              <xsl:template match="xhtml:table">
                <xsl:for-each select="xhtml:tr">
                  <xsl:if test="position()!=1">
                    <!-- Fully scope quote with the response XML schema file -->
                    <pflResp:Quote>
                    <!-- Fully scope symbol also -->
                      <pflResp:Symbol>
                        <xsl:value-of select="xhtml:td[1]/xhtml:a"/>
                      </pflResp:Symbol>
                      <pflResp:Time>
                        <xsl:value-of select="xhtml:td[2]"/>
                      </pflResp:Time>
                      <pflResp:Price>
                        <xsl:value-of select="xhtml:td[3]/xhtml:b"/>
                      </pflResp:Price>
                      <pflResp:Change>
                        <xsl:value-of select="xhtml:td[5]"/>
                      </pflResp:Change>
                      <pflResp:Volume>
                        <xsl:value-of select="xhtml:td[6]"/>
                      </pflResp:Volume>
                    </pflResp:Quote>
                  </xsl:if>
                </xsl:for-each>
              </xsl:template>
            </xsl:stylesheet>
          </xoParams:XSLT>
        </xoParams:XSLT_OA_PARAMS>
      </sd:PARAMETERS>
    </sd:ADAPTOR>
</sd:OUTPUT>
```

The DSXSLTOutputAdaptor specified first, converts the returned HTML into a more XML-compliant XHTML format. It then applies the supplied XSL stylesheet to that XHTML document to form an XML document that conforms to the response XML schema file that you previously defined in Section 6.1.5.

Close the service body and the service descriptor elements as shown in Example 6–16.

Example 6–16 Close the Service Body and Service Descriptor Elements

```
  </sd:SERVICE_BODY>
</sd:SERVICE_DESCRIPTOR>
```

6.1.7 Testing the Execution of Your Service

After constructing the service package and editing the service descriptor, do the following to test the execution of your service:

1. Run the DSAdmin utility. Use the DSAdmin utility to:
   a. Add the categories by following the instructions in Section 3.2.1.
   b. Register the service by pointing to the `demo/services/myService` directory on UNIX systems or `demo\services\myService` directory on Windows NT systems as described in Section 3.2.2.

   Note: Anytime you make a change to any service related file, you must reregister that service package using the DSAdmin utility Reregister command.

2. Build a sample service request definition and a sample service response definition following the examples described in Example 6–7 and Example 6–9.
3. Use the DSAdmin utility to execute the service. See Section 3.4 for more information.

You can turn the execution output level to *trace* by selecting **Prop (P)** at the top-level menu, and then selecting **Change (C)** to change the debug output levels. Finally, select **TRACE (3)** to turn it to the trace level, so you can see every step of the execution flow.

This completes the description of the steps needed to create, register, and test a simple HTTP service.

To create more advanced services, see Section 6.2 through Section 6.5.

6.2 Creating Advanced Services -- Service Package

The service package is structured as a local directory containing a set of files with the following structures:

- A `MANIFEST` file pointing to the service descriptor

- The service descriptor XML file, and other XML files it points to, including classification, provider information (organization and contacts), input or output schemas, and so forth
- A jar file containing all Java classes and stylesheets needed by the service

The `MANIFEST` file is expected to be found in the root directory of the package, and with the name MANIFEST (uppercase, case-sensitive on Solaris systems; initial uppercase, non case-sensitive on Windows NT systems). The `MANIFEST` file is a text file where the first non-empty line should specify a URL link to the service descriptor. If a link starts with a slash (/), it indicates the link is an absolute link with respect to the root of the current service package. The root is interpreted to be the root of the directory structure of the service package. The `MANIFEST` file must end with a new line.

Note: The *<ORACLE_HOME>*`/ds/etc/xsd` directory on UNIX systems or the *<ORACLE_HOME>*`\ds\etc\xsd` directory on Windows NT systems contains the XML schema for the service descriptor and supplied adaptors. See the files in this directory for more information.

6.3 Creating Advanced Services -- Service Descriptor

A service package is modeled through an XML document called a service descriptor that provides a centralized source of description for the service. A service is defined by a multitude of logical components, all of which are specified in the service descriptor or in other documents to which the descriptor refers. There are two sections of the service descriptor:

- Service header: Describes the high-level behavior of the service
- Service body: Describes the implementation details of the service

These descriptor sections, described in Section 6.3.1 and Section 6.3.2 correspond to XML elements with the same names in the service descriptor. For service descriptor examples, refer to the supplied sample services under the *<ORACLE_HOME>*`/ds/demo/services` directory on UNIX systems or the *<ORACLE_HOME>*`\ds\demo\services` directory on Windows NT systems.

6.3.1 Service Header Section

The service header section contains high-level behavior descriptions of the service. For the most part, information specified in the service header section is descriptive

and non-interpretive and primarily for browsing and documentation purposes. The exceptions are the service identifier specification described in Section 6.3.1.1 and the service interface specification described in Section 6.3.1.5.

6.3.1.1 Naming Specification

The service header section has naming information that contains a globally unique identifier for the services, as well as short and long descriptions of what the service does. Each service has a unique name specified using the universal resource name (URN) conventions. Example 6–17 shows a sample naming specification.

Example 6–17 Sample Naming Specification

```
<!-- Naming specification provides identification information about the
     service. Among the elements, ID is the important one, which
     serves as unique identification. It must be a URN. -->
<sd:NAMING>
  <sd:ID>urn:com.foobar:service_name</sd:ID>
  <sd:NAME>Put a human-readable name here.</sd:NAME>
  <sd:DESCRIPTION>Some description about the service.</sd:DESCRIPTION>
</sd:NAMING>
```

6.3.1.2 Package Specification

The service header section includes package information with version specifications and pointers to how and where the service update is to be performed. Coupled with support contacts from the service provider information section (see Section 6.3.1.3), this information is critical for service maintenance. Example 6–18 shows a sample package specification.

Example 6–18 Sample Package Specification

```
<!-- Package provides version information, update locations, and
     binary resource specifications. -->
<sd:PACKAGE>
  <sd:VERSION>1.0</sd:VERSION>
  <sd:RELEASEDATE>25-MAR-2000</sd:RELEASEDATE>
  <sd:UPDATEURL>http://www.foobar.com/dServices/svc.zip</sd:UPDATEURL>
  <sd:BINARY_RESOURCES>
    <!-- JAR_POINTER is used only in the special case that you
         have custom Java classes. Skip the whole JAR_POINTER
         section if no custom Java classes are needed. -->
    <sd:JAR_POINTER
      xlink:href="/www.foobar.com/dServices/dummy.jar" />
    <!-- EXCEPTIONS is the section where the resource bundle for
```

```
        custom exceptions can be specified. If the exceptions
        do not rely on custom resource bundles, the whole EXCEPTIONS
        section can be skipped. -->
    <sd:EXCEPTIONS>
      <sd:EXCEPTION_MSG_BUNDLE>com.foobar.Bundle</sd:EXCEPTION_MSG_BUNDLE>
    </sd:EXCEPTIONS>
  </sd:BINARY_RESOURCES>
</sd:PACKAGE>
```

6.3.1.3 Service Provider Specification -- Organization and Contacts

This service header section includes high-level information about the service provider, including the service provider's company name, copyright information, and company URL. Detailed information includes contacts for support and URLs for logos. This information is provided in the form of an X-Link that points to another XML document in the service package. Example 6–19 shows a sample service provider specification.

Example 6–19 Sample Service Provider Specification

```
<sd:PROVIDER>
  <!-- The ORGANIZATION section is a mandatory document that gives information
       about the service provider. For a quick start, it can be filled
       with dummy data. -->
  <sd:ORGANIZATION
    xlink:href="/www.foobar.com/dServices/foobar_org.xml"/>
  <!-- Each ORGANIZATION section can be associated with zero or more
       contact documents.-->
  <sd:CONTACTS>
    <sd:CONTACT xlink:href="/www.foobar.com/dServices/contact.xml"/>
  </sd:CONTACTS>
</sd:PROVIDER>
```

6.3.1.4 Deployment Specification -- Classification and Caching

This service header section includes a set of deployment properties that includes suggestions from the service provider to aid the service administrator during registration time. These suggestions include classification guidelines with hierarchical categories, as well as flat keywords and recommendations of caching parameters. This information is also provided in the form of an X-Link that points to another XML document specifying the classification schemes. The values specified here are only suggestions to a service administrator during service registration. The values stored in the service registry could be different from the

values specified in the service descriptor. Example 6–20 shows a sample deployment specification.

Example 6–20 Sample Deployment Specification

```
<sd:DEPLOYMENT>
  <!-- Follow the convention in path name within the zip file. -->
  <sd:CLASSIFICATION
    xlink:href="/www.oanda.com/dServices/currency/class.xml"/>
  <sd:CACHING>
    <sd:MAX_AGE>300</MAX_AGE>
    <sd:SESSION_PRIVATE>false</sd:SESSION_PRIVATE>
    <sd:USE_PROTOCOL>false</sd:USE_PROTOCOL>
  </sd:CACHING>
</sd:DEPLOYMENT>
```

See Section 7.3 for more information about service response caching.

6.3.1.5 Service Interface Specification -- Request and Response Definitions

The service header allows for the definition of an interface characterized by the schema specifications of its input, output, and exceptions. The specifications are dispersed in external XML schema documents. The location of the XML schema document file is specified by URLs, when a relative URL is used, that is, relative to the service package submitted by the service providers. By specifying these schemas, the service provider enforces the syntax in which service consumer applications send requests to it, as well as the way in which it provides the responses. The validation will be done in the Dynamic Services engine when a service consumer application sends a request, before the service provider is contacted.

The service provider can also suggest a name for the interface, which is a deployment option and can be overwritten by the service administrator. Any new service that conforms to the same service interface must provide the same input/output (not necessarily the exception) definition. The Dynamic Services engine also exposes to service consumer applications the capability to search for services by interface. Two services that conform to the same interface are considered compatible services, a concept useful for failover.

Note: To facilitate the development of code that works with Dynamic Services, class generators can be used to create Java classes that correspond to the request/response XML schema files.

Example 6–21 shows a sample service interface specification.

Example 6–21 Sample Service Interface Specification

```
<sd:INTERFACE>
  <sd:NAME>FoobarTemplate</sd:NAME>
  <sd:INPUT_SCHEMA xlink:href="/www.foobar.com/dServices/fb_req.xsd"/>
  <sd:OUTPUT_SCHEMA xlink:href="/www.foobar.com/dServices/fb_resp.xsd"/>
</sd:INTERFACE>
```

6.3.2 Service Body Section

The service body section contains more detailed descriptions and information used by the Dynamic Services engine at service execution time. Specifically, its sections are specifications (including adaptors) on the input, protocol, execution, and output, where:

- Input deals with the handling of the submitted service request
- Protocol adapts the XML service request to the communication protocol used by the remote service provider
- Execution determines the execution flow of a service
- Output transforms the raw response returned by the remote service provider into a service XML response

Figure 6–1 shows a sample service execution and the roles of the input, protocol, and output adaptors, and the flow of information.

Figure 6–1 Sample Service Execution Showing the Role of the Input, Protocol, and Output Specifications as Specified Adaptors

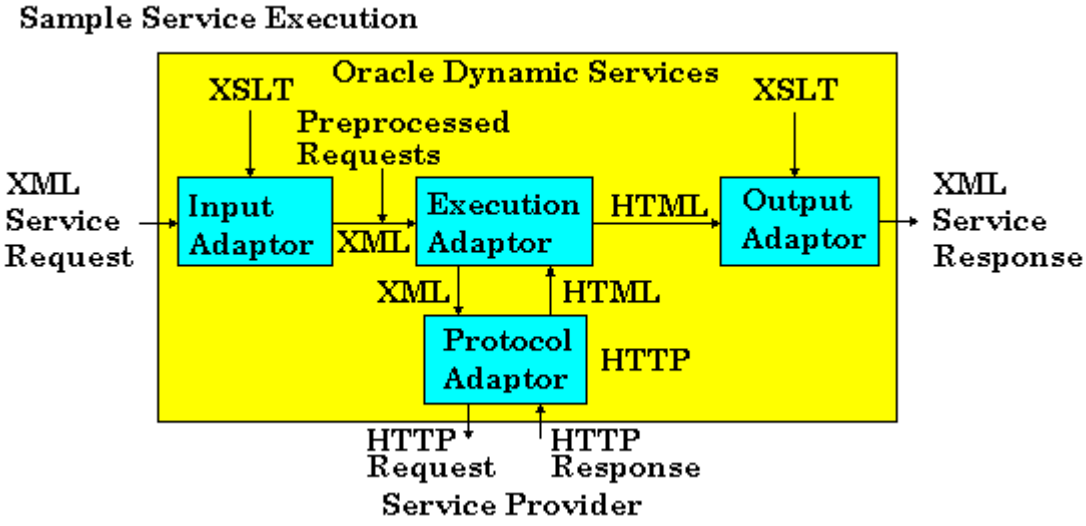
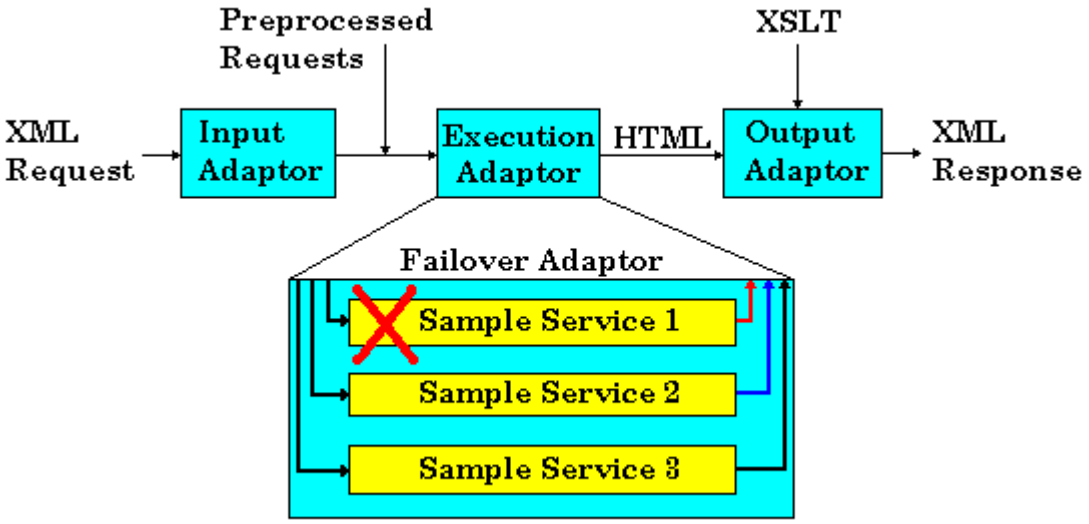


Figure 6–2 shows a sample execution adaptor and the role the execution adaptor plays in identifying the way in which one or more sample services is to be executed. In this case, an execution adaptor would specify in its execution flow logic how and why a set of one or more sample services is to be executed. For example, a failover execution adaptor would specify the preferred order of execution of the sample services from its list of compatible services in the event that one or more services failed to execute. In this figure, sample service 1 fails to execute, thus sample service 2 is executed; meanwhile sample service 3 is ready for execution in the event that sample service 2 fails to execute.

Figure 6–2 Sample Execution Adaptor



The service provider can specify the needed adaptor for each of these layers. A set of pre-built, customizable adaptors is supplied by Oracle Dynamic Services. See Section 6.4 for a description of these adaptors.

6.3.2.1 Input Handling and Adaptor Specifications

The service body section has input specifications that provide a list of necessary as well as optional processing steps for the request that is submitted by the service consumer application. This includes the following input specifications:

- Namespaces
- Alias directives
- Input adaptor
- Rendering directives

Section 6.3.2.1.1 through Section 6.3.2.1.4 describe each of these input specifications.

6.3.2.1.1 Input: Namespaces A list of namespaces with their prefixes can be specified before specifying the aliases. The prefixes can be used in the aliases specification to

build the XPaths pointing to where the data is. If no namespaces are needed, this item can be skipped. Example 6–22 shows a sample namespaces specification.

Example 6–22 Sample Namespaces Specification

```
<sd:NAMESPACES>
  <sd:NAMESPACE>
    <sd:PREFIX>fb</sd:PREFIX>
    <sd:VALUE>http://www.foobar.com/foobar/Request</sd:VALUE>
  </sd:NAMESPACE>
</sd:NAMESPACES>
```

6.3.2.1.2 Input: Aliases Directives Service providers may specify alias directives. Aliases are used to create a map that can translate the parameters embedded in the XML service request document to actual parameters needed by the communication protocol of the service. For example, for HTTP, a service provider can specify an XPath for the request XML document addressing an element that represents one of the HTTP request parameters to be sent to the HTTP server. There are currently two possible ways of specifying an alias:

- Supply an XPath, which at service execution time, is applied to the XML service request file and used to extract either the value of the node pointed to by the XPath, or the XML fragment for which this node is the root.
- Supply a service consumer application profile property and optionally its modifier. The corresponding value is fetched dynamically at execution time according to the identity of the service consumer application. See Section 7.1 for more information about how to manage properties.

Example 6–23 shows a sample aliases specification, the first using the XPath approach for a value, using the namespace described previously; the second using the XPath approach for a document fragment; and the last using the profile property approach.

This sample uses the `fb` prefix and the namespace, `http://www.foobar.com/foobar/Request` described in Section 6.3.2.1.1.

Example 6–23 Sample Aliases Specification

```
<sd:ALIASES>
  <sd:ALIAS>
    <sd:NAME>AccountNumber</sd:NAME>
    <!-- Indicates that the value is obtained dynamically from the user
         request following the namespace specifed by fb -->
    <sd:VALUE>{@xpath:value=/fb:foobarReq/fb:param1}</sd:VALUE>
```

```
  </sd:ALIAS>
    <sd:ALIAS>
    <sd:NAME>SomeFragment</sd:NAME>
    <!-- Indicates that the value is obtained dynamically from the user
         request following the namespace specifed by fb -->
    <sd:VALUE>{@xpath:fragment=/fb:foobarReq/fb:frag1}</sd:VALUE>
  </sd:ALIAS>
  <sd:ALIAS>
    <sd:NAME>Password</sd:NAME>
    <!-- Indicates that the value of the alias should be retrieved as a
         property from the service consumer application profile registry
         and that the property name is foobarProp1. -->
    <sd:VALUE>{@dscr:property=foobarProp1}</sd:VALUE>
  </sd:ALIAS>
</sd:ALIASES>
```

6.3.2.1.3 Input: Adaptor Specification The input adaptor specification can optionally specify an adaptor that further processes the service request before sending it to the service provider. Examples of such processing include semantic or higher-level validation of the request. This input adaptor specification is a fully qualified name of the class implementing the `oracle.ds.engine.InputAdaptor` Java interface that handles the processing. For the specified adaptor, the service provider has the option of specifying some adaptor-specific parameters in the PARAMETERS element under the adaptor specification that are validated at service registration time, and interpreted at runtime by the adaptor. The parameters are *opaque* to the service descriptor parser and service registry, and must be in XML syntax.

6.3.2.1.4 Input: Rendering Directives Under normal execution flow, the request XML that the service consumer application submits or sends to the Dynamic Services engine is validated with the input XML schema file that is specified previously in the header. However, a service provider can optionally supply some form of schema mapping specifications (for example, through an XSL transformation) that could map this input XML schema file to a presentation form such as HTML or Wireless Markup Language (WML). As a result, the service consumer application can easily provide to its clients a way to submit service requests, for applications that have an HTML or WML interface.

The Dynamic Services engine is not responsible for the rendering: all that the engine is responsible for is the capabilities to store and retrieve the mapping. The Dynamic Services engine provides only the mapping(s) of the transformation. The actual transformation is done on the service consumer application side by the client toolkit. If you map the schema to an HTML form, the service consumer applications can use the mapping to render the input schema to an HTML form for their Web

application. The service consumer application can then transform the HTTP requests back to an XML document that conforms to the request XML schema file specified by the service provider. Finally, the request XML schema file is sent to the Dynamic Services engine formulating the service request.

6.3.2.2 Protocol Adaptor Specification

The protocol adaptor specification identifies the way that the Dynamic Services engine accesses the underlying service. For example, a service may be accessed through the HTTP protocol, while another service may be accessed through the JDBC protocol. This protocol adaptor specification is a fully qualified name of the class implementing the `oracle.ds.engine.ProtocolAdaptor` interface. The interface handles the communication to the underlying service. This class name is found either in the service package given by the service provider during registration, or in the set of libraries that the Dynamic Services engine provides.

A driver specification can make sure that a certain class is in the classpath for the adaptor to function properly.

Finally, for the identified adaptor, the service provider has the option of specifying some adaptor-specific parameters in the PARAMETERS element in the adaptor specification, which are validated at service registration time and interpreted at runtime by the adaptor. For example, for HTTP, the adaptor may specify the HTTP method used and the URL that does the actual servicing. These parameters are *opaque* to the service descriptor parser and service registry, and must be in XML syntax.

Example 6–24 shows a sample protocol adaptor specification using the HTTPS protocol adaptor.

Example 6–24 Sample HTTPS Protocol Adaptor Specification

```
<sd:PROTOCOL>
  <sd:ADAPTOR>
    <sd:NAME>oracle.ds.engine.pa.http.DSHTTPSProtocolAdaptor</sd:NAME>
    <sd:DRIVER>com.sun.net.ssl.HttpsURLConnection</sd:DRIVER>
    <sd:PARAMETERS>
      <!-- Adaptor-specific parameters -->
    </sd:PARAMETERS>
  </sd:ADAPTOR>
</sd:PROTOCOL>
```

6.3.2.3 Execution Adaptor Specification

The execution adaptor specification identifies the way in which the service has to be executed. Its responsibility is to receive the request XML schema file and return the response from the underlying service provider.

The default execution adaptor is a standard simple adaptor that follows the path described previously. There can also be complex or compound execution adaptors that aggregate several services, such as in the International Portfolio example.

This execution adaptor specification is a fully qualified class name of a class implementing the `oracle.ds.engine.ExecutionAdaptor` interface, which performs the execution. For the identified adaptor, the service provider has the option of specifying some adaptor-specific parameters in the PARAMETERS element in the adaptor specification, which are validated at service registration time and interpreted at runtime by the adaptor.

The result of the execution adaptor is the response from the service. If the service is a simple service, the response will be in the native format of the service provider. For example, for a Web-based service, the response may be in HTML format. If the service is a compound service, the response will be a structured service response.

Usually, if the service is a simple service, a service provider will use the default prepackaged simple adaptor. For complex and compound service execution, the service provider can use the supplied compound execution adaptor or DSFailOverExecutionAdaptor or DSConditionalExecutionAdaptor to greater advantage.

6.3.2.4 Output Handling and Adaptor Specification

The output specification specifies the list of necessary as well as optional processing to produce the service response to the service consumer application. This includes the following output specifications:

- Output adaptor
- Rendering directives

Section 6.3.2.4.1 and Section 6.3.2.4.2 describe each of these output sections.

6.3.2.4.1 Output: Adaptor Specification The output adaptor specification can specify an output adaptor to be used to transform the output returned by the execution adaptor. That output is transformed into an XML document compliant with the output XML schema file specified in the service interface. This output adaptor specification is a fully qualified name of the class implementing the `oracle.ds.engine.OutputAdaptor` interface, which handles the

transformation. For the identified output adaptor, the service provider has the option of specifying some adaptor-specific parameters in the PARAMETERS element in the adaptor specification, which are validated at service registration time and interpreted at runtime by the adaptor. These parameters are *opaque* to the service descriptor parser and service registry, and must be in XML syntax.

Usually, for simple services, service providers will use either of the prepackaged adaptors, such as the XSLT adaptor. For compound services, service providers can use no adaptor because the response from the execution adaptor is often in the proper format prescribed by the output XML schema file.

6.3.2.4.2 Output: Rendering Directives As far as the service execution flow is concerned, the output specification is the final step. However, Dynamic Services also provides additional mechanisms for the service provider to optionally specify mappings (for example, in the form of XSL transforms) that map this response XML file to other forms, such as HTML or WML. Service consumer applications, rather than the Dynamic Services engine, are responsible for making use of the transformation to produce the desired output. Dynamic Services merely provides a means to store it and make it accessible from the service consumer application.

6.4 Creating Advanced Services -- Description of Supplied Adaptors

Table 6–1 describes the complete set of supplied adaptors provided by Oracle Dynamic Services.

Table 6–1 Adaptors Supplied by Oracle Dynamic Services

Adaptor name	Type
oracle.ds.engine.ioa.DSXSLTInputAdaptor	Input
oracle.ds.engine.pa.http.DSHTTPProtocolAdaptor	Protocol
oracle.ds.engine.pa.http.DSHTTPSProtocolAdaptor	Protocol
oracle.ds.engine.pa.jdbc.DSJDBCProtocolAdaptor	Protocol
oracle.ds.engine.pa.smtp.DSSMTPProtocolAdaptor	Protocol
oracle.ds.engine.ea.DSFailOverExecutionAdaptor	Execution
oracle.ds.engine.ea.compound.DSCompoundServiceExecutionAdaptor	Execution
oracle.ds.engine.ea.DSConditionalExecutionAdaptor	Execution
oracle.ds.engine.ioa.DSXSLTOutputAdaptor	Output

Section 6.4.1 through Section 6.4.4 offer a more detailed description of the adaptors supplied by Oracle Dynamic Services.

6.4.1 Input Adaptor

Section 6.4.1.1 describes the input adaptor supplied by Oracle Dynamic Services.

6.4.1.1 oracle.ds.engine.ioa.DSXSLTInputAdaptor

DSXSLTInputAdaptor applies an XSLT transformation to the incoming requests and returns the transformed request as a result. In order to use this adaptor, the INPUT/ADAPTOR/NAME must be:
`oracle.ds.engine.ioa.DSXSLTInputAdaptor`

The INPUT/ADAPTOR/PARAMETERS element in the service descriptor must contain what is shown in Example 6–25.

Example 6–25 Sample XSL Stylesheet Information

```
<xiParams:XSLT_IA_PARAMS
  xmlns:xiParams="http://www.oracle.com/ds/2000/XSLT_IA_PARAMS">
  <xiParams:XSLT>
    <!-- The XSL Stylesheet -->
  </xiParams:XSLT>
</xiParams:XSLT_IA_PARAMS>
```

The specified XSL stylesheet can use the aliases defined as `xsl` variables in the two ways shown in Example 6–26.

Example 6–26 Sample Aliases Defined as XSL Variables

When the value of the alias is used as an XPath expression:

```
<xsl:variable name="myvar"select = "{@aliasName}"/>
```

Or, when the value of the alias is used as a string literal:

```
<xsl:variable name="myvar">{@aliasName}</xsl:variable>
```

In the DSXSLTInputAdaptor, there is an option of bringing in other service descriptors before applying the stylesheet. This can be done using an attribute of the xiParams:XSLT element called `applyWithServiceDescriptor`. Refer to the notifier event monitor service that comes with the Oracle Dynamic Services installation. For more examples of the input adaptor specification for DSXSLTInputAdaptor, refer to the YahooPortfolio service.

6.4.2 Protocol Adaptors

Section 6.4.2.1 through Section 6.4.2.4 describe the protocol adaptors supplied by Oracle Dynamic Services.

6.4.2.1 oracle.ds.engine.pa.http.DSHTTPProtocolAdaptor

DSHTTPProtocolAdaptor processes the HTTP/1.0 specification, sets up the HTTP/1.0 connection, and returns a handler to process the raw output from the service provider.

The parameters and other details needed to execute DSHTTPProtocolAdaptor are defined in the service description, in the section SERVICE_BODY/PROTOCOL/ADAPTOR as shown in Example 6–27.

Example 6–27 Sample HTTP Protocol Adaptor Specification

```
<sd:SERVICE_BODY>
...
  <sd:PROTOCOL>
    <sd:ADAPTOR>
      <sd:NAME>oracle.ds.engine.pa.http.DSHTTPProtocolAdaptor</sd:NAME>
      <sd:DRIVER>java.net.URLConnection</sd:DRIVER>
      <sd:PARAMETERS>
        <!-- Each protocol adaptor has a namespace where all the
             following elements and/or attributes are defined: -->
        <hpParams:HTTP_PA_PARAMS
          xmlns:hpParams="http://www.oracle.com/ds/2000/HTTP_PA_PARAMS">
          <hpParams:Method>GET</hpParams:Method>
          <hpParams:URL>www.oanda.com/converter/classic</hpParams:URL>
          <hpParams:QueryStringParameters>
            <hpParams:QueryStringParameter
              name= "pname1">{@pvalue1}</hpParams:QueryStringParameter>
            <hpParams:QueryStringParameter
              name= "pname2">{@pvalue2}</hpParams:QueryStringParameter>
          </hpParams:QueryStringParameters>
          <hpParams:RequestHeaders>
            <hpParams:RequestHeader name="hdr1">{@val1}</hpParams:RequestHeader>
            <hpParams:RequestHeader name="hdr2">{@val2}</hpParams:RequestHeader>
          </hpParams:RequestHeaders>
        </hpParams:HTTP_PA_PARAMS>
      </sd:PARAMETERS>
    </sd:ADAPTOR>
  </sd:PROTOCOL>
```

This specification consists of two parts. One is a generic part, the other is specific for the given adaptor. The first part includes <NAME> and <DRIVER> elements, which indicate the class names of the specific HTTP adaptor and the handler to process the response from the HTTP resource. The second part, bounded by <PARAMETERS>, is for the specific adaptor; in this case, it must fit the requirements of DSHTTPProtocolAdaptor.

The DSHTTPProtocolAdaptor parameters consist of mandatory and optional elements. The mandatory elements include <Method> and <URL>. <Method> must be one of the three options: GET, POST, or HEAD. The optional elements are <QueryString> and <Authorization>.

In some cases, there may be one or more parameters for the HTTP query string. Each <QueryStringParameter> element within <QueryStringParameters> element defines one parameter. The parameter name is specified as the "name" attribute of the <QueryStringParameter> element, while the parameter value is the element value of <QueryStringParameter>. The element value <QueryStringParameter> may be an alias, which is resolved according to what has been defined in the section SERVICE_BODY / INPUT / ALIASES.

The <RequestHeaders> element is optional in the specification; it is useful for manually setting HTTP request headers in the request. In the example, the two request headers that are set will be used every time an HTTP request is made. The request header name is specified with the "name" attribute of <RequestHeader>, while the request header value is the element value of <RequestHeader>.

Another optional element is <Authorization>, which is useful for some secured Web sites that require the user's login name and password. According to the HTTP / 1.0 specification, the content of <Authorization> can be defined in one of two possible structures. The first one puts the login name and password as a single string, while the second one separates them. In both cases, the login name and password are encrypted in Base64. The login name and password can be aliases that refer to other sources as shown in Example 6–28. For more information, refer to the HTTP 1.0 specification Web site listed in Appendix A.

Example 6–28 Sample Login and Password Aliases in the Authorization Specification

```
<hpParams:Authorization>
      <hpParams:EncodedString>encodedloginpasswd</hpParams:EncodedString>
</hpParams:Authorization>

<hpParams:Authorization>
  <hpParams:Credential>
    <hpParams:Username>encodedusername</hpParams:Username>
```

```
    <hpParams:Password>encodedpassword</hpParams:Password>
  </hpParams:Credential>
</hpParams:Authorization>
```

DSHTTPProtocolAdaptor also supports cookies (see Section 5.1.6). The lifetime of cookies lasts only as long as a service consumer application session.

6.4.2.2 oracle.ds.engine.pa.http.DSHTTPSProtocolAdaptor

This release also provides an HTTPS protocol adaptor. Syntactically it is identical to DSHTTPProtocolAdaptor.

The implementation is based on Sun Microsystems JSSE 1.0.2 global reference implementation.

6.4.2.3 oracle.ds.engine.pa.jdbc.DSJDBCProtocolAdaptor

DSJDBCProtocolAdaptor allows services to interface with an Oracle database by creating a row response from the execution of a SQL statement or a PL/SQL stored procedure. The DSJDBCProtocolAdaptor uses the Oracle XSQL pages technology to express SQL or PL/SQL operations in XML. The parameters supplied in the adaptor are used to define the database connections to be used as well as the XSQL page to be executed. For more information about XSQL pages, see the Oracle XSQL Servlet documentation available on Oracle Technology Network.

Example 6–29 shows the parameters and other details to execute DSJDBCProtocolAdaptor as defined in the service descriptor, in the section SERVICE_BODY/PROTOCOL/ADAPTOR.

Example 6–29 Sample JDBC Protocol Adaptor Specification

```
<sd:SERVICE_BODY>
...
  <sd:PROTOCOL>
    <sd:ADAPTOR>
      <sd:NAME>oracle.ds.engine.pa.jdbc.DSJDBCProtocolAdaptor</sd:NAME>
      <sd:DRIVER>oracle.jdbc.driver.OracleDriver</sd:DRIVER>
      <sd:PARAMETERS>
        <!-- Each protocol adaptor has a namespace where all the
              following elements and/or attributes are defined: -->
        <jpParams:JDBC_PA_PARAMS
          xmlns:jpParams="http://www.oracle.com/ds/2000/JDBC_PA_PARAMS"
          xmlns:xsql= "urn:oracle-xsql">

        <jpParams:connectiondefs>
```

```
          <jpParams:connection name="demo">
          <jpParams:username>{@username}</jpParams:username>
          <jpParams:password>{@password}</jpParams:password>
          <jpParams:dburl>jdbc:oracle:thin:@hostname:port:sid</jpParams:dburl>
          <jpParams:driver>oracle.jdbc.driver.OracleDriver</jpParams:driver>
          <jpParams:autocommit>true</jpParams:autocommit>
        </jpParams:connection>
        </jpParams:connectiondefs>

        <jpParams:page connection="demo">
        <xsql:query> select ename from emp </xsql:query>
        </jpParams:page>


          </jpParams:JDBC_PA_PARAMS>
        </sd:PARAMETERS>
      </sd:ADAPTOR>
    </sd:PROTOCOL>
```

These DSJDBCProtocolAdaptor parameters consist of two parts: the first one defines the JDBC connections necessary for processing the XSQL page, while the second one defines the XSQL page to be processed. Because the implementation of DSJDBCProtocolAdaptor uses the Oracle XSQL package, the xsql namespace must be declared in the JDBC_PA_PARAMS elements to correctly scope the XSQL page elements. The use of <username>, <password>,<dburl>, <driver>, and <autocommit> are consistent with those defined in Oracle JDBC Driver 2.0.

As defined by the XSQL syntax, any query statement is quoted within the pair of <xsql:query>, while other DML statements are bounded by <xsql:dml> pairs. The statements may contain some aliases, which are resolved according to what has been defined in the SERVICE_BODY/INPUT/ALIASES section.

In general, all the ALIASES defined for the service are supplied as XSQL parameters to the XSQL page processor. DBService is a sample service to illustrate how to execute any actions to a database.

If additional resources need to be accessed by the XSQL page (for example another XSQL page, an XSL stylesheet, or an XML document), they can be bundled in a jar file packaged as a binary resource for the service. The location of the jar file containing the resource has to be specified as a service binary resource using the BINARY_RESOURCE/JAR_POINTER element in the package section of the service header. The path specified to access those resources will be used to load them from the supplied jar file.

When a service using DSJDBCProtocolAdaptor is executed within a service consumer (application) session, the JDBC connection (identified by its name) is reused within the session. Also, if within a session, two connections are defined with the same name but their connection strings (identified as user name, password, and dburl) do not match, an exception will be raised.

At session closing time, the JDBC connection is rolled back. To commit any updates, a service must explicitly make a commit() call, or set autocommit to be true. The behavior is analogous to discarding cookies for an HTTP connection. It is the responsibility of the service consumer (application) to close any session that it created so that the associated resource will be released.

See the DBService sample service package for more information.

6.4.2.4 oracle.ds.engine.pa.smtp.DSSMTPProtocolAdaptor

DSSMTPProtocolAdaptor is the protocol adaptor used for service providers that use SMTP as an underlying service access mechanism. Such services can include simple mail sending services that get invoked upon the occurrence of an error.

The parameters and other details needed to execute DSSMTPProtocolAdaptor are defined in the service descriptor, in the section SERVICE_BODY/PROTOCOL/ADAPTOR, as shown in Example 6–30. They include information such as the host name, port number of the SMTP server, the from, to, cc, bcc, and subject fields of an e-mail message, additional message headers, and finally a message body. This is an example that is taken from the notifier service that comes with the installation package.

Example 6–30 Sample SMTP Protocol Adaptor Specification

```
<sd:SERVICE_BODY>
...
  <sd:PROTOCOL>
      <sd:ADAPTOR>
        <sd:NAME>oracle.ds.engine.pa.smtp.DSSMTPProtocolAdaptor</sd:NAME>
        <sd:DRIVER>java.net.Socket</sd:DRIVER>
        <sd:PARAMETERS>
          <!-- Predefined XML schema of the SMTP protocol adaptor parameters -->
          <spParams:SMTP_PA_PARAMS
            xmlns:spParams="http://www.xyz.com/ds/2000/SMTP_PA_PARAMS">
            <spParams:Host>server1.xyzcorp.com</spParams:Host>
            <spParams:Port>25</spParams:Port>
            <spParams:From>group@server2.xyz.com</spParams:From>
            <!-- Notice how aliases can be used in each of these fields. -->
            <spParams:To>{@To}</spParams:To>
```

```
            <spParams:cc>{@cc}</spParams:cc>
            <spParams:Subject>
              Notification for svc: {@EvtSvc} Op: {@EvtType} Stat: {@EvtStat}
            </spParams:Subject>

            <spParams:MsgHeaders>
              <spParams:MsgHeader name="Mime-Version">1.0</spParams:MsgHeader>
              <spParams:MsgHeader
                name="Content-Type">text/html;charset="us-ascii"
              </spParams:MsgHeader>
            </spParams:MsgHeaders>

            <spParams:MsgBody>
              <html>
                <title>Notification for service {@EvtService}.
                       Operation: {@EvtType} Status: {@EvtStatus}</title>
                <body>
                  <H3>Service:     <B>{@EvtService}</B></H3>
                  <H3>Consumer:    <B>{@EvtConsumer}</B></H3>
                  <H3>TimeStamp:   <B>{@EvtTimeStamp}</B></H3>
                  <H3>Operation:   <B>{@EvtType}</B></H3>
                  <H3>Status:      <B>
                    <font color="red">{@EvtStatus}</font></B>
                  </H3>
                  <H3>Description: <B>{@EvtDescription}</B></H3>
                  <P></P>
                  <H3>Message</H3>
                  <PRE>{@EvtBody}</PRE>
                </body>
              </html>
            </spParams:MsgBody>
          </spParams:SMTP_PA_PARAMS>
        </sd:PARAMETERS>
      </sd:ADAPTOR>
    </sd:PROTOCOL>
```

In Example 6–30, the message body actually is an HTML document that displays the status of a certain operation.

6.4.3 Execution Adaptors

Section 6.4.3.1 through Section 6.4.3.3 describe the execution adaptors supplied by Oracle Dynamic Services.

6.4.3.1 oracle.ds.engine.ea.DSFailOverExecutionAdaptor

DSFailOverExecutionAdaptor takes as parameters an ordered list of compatible services, which means the services respond to the same service interface. At execution time, the failover execution adaptor tries to execute the first one in the list; if it fails, it moves to the second list item, and so on, until it finds a service that executes with no exception. If none succeeds, an exception is raised. Example 6–31 shows a sample adaptor specification for a failover service taken from the FailOverPortfolio service.

Example 6–31 Sample Failover Adaptor Specification

```
<sd:EXECUTION>
  <sd:ADAPTOR>
    <sd:NAME>oracle.ds.engine.ea.DSFailOverExecutionAdaptor</NAME>
    <sd:PARAMETERS>
      <feParams:FAILOVER_EA_PARAMS
        xmlns:feParams="http://www.oracle.com/ds/2000/FAILOVER_EA_PARAMS">
        <feParams:execute priority="0">
          urn:com.yahoo:finance.portfolio_fails
        </feParams:execute>
        <feParams:execute priority="1">
          urn:com.cnnfn:finance.portfolio03
        </feParams:execute>
      </feParams:FAILOVER_EA_PARAMS>
    </sd:PARAMETERS>
  </sd:ADAPTOR>
</sd:EXECUTION>
```

6.4.3.2 oracle.ds.engine.ea.compound.DSCompoundServiceExecutionAdaptor

DSCompoundServiceExecutionAdaptor controls the execution of compound services. Example 6–32 shows the XML portion of a compound service, execution adaptor specification.

Example 6–32 Sample Compound Service Specification

```
<sd:EXECUTION>
  <sd:ADAPTOR>

<sd:NAME>oracle.ds.engine.ea.compound.DSCompoundServiceExecutionAdaptor</NAME>
    <sd:PARAMETERS>
      <ceParams:COMPOUND_EA_PARAMS
        xmlns:ceParams="http://www.oracle.com/ds/2000/COMPOUND_EA_PARAMS">
```

Compound services let you encapsulate the execution of a multitude of services by combining them into a directed graph of service executions. Each node of the graph is identified as a CompoundEAModule. There are four possible types of modules that can be used in the graph. Each of the modules is designed as a JavaBean with exposed properties. Those properties are set at compound service design time (probably through service provider design tools), and persist through runtime, when they are used to control the execution.

DSCompoundServiceExecutionAdaptor coordinates the execution of the modules according to the graph specifications, triggering the module executions through JavaBeans events. The following subsections describe the four available modules and their properties.

oracle.ds.engine.ea.compound.ServiceExecution

This module executes one service. It accepts an array of messages, interpreting them as requests, and produces another array of messages composed of responses returned by the service execution(s). There are two possible syntax forms for its properties:

- executeSingleRequest

  With this option for the properties, the module takes in a message index number as an attribute -- a request event can contain a list of requests -- and takes the ID of the service to be executed, as an element value. Only one execution is performed using the one selected request message, and the service executed is the one specified by the ID with that request. Example 6–33 shows a sample ServiceExecution module specification with the executeSingleRequest property option.

Example 6–33 Sample Service Execution Module with the executeSingleRequest Property

```
<ceParams:Module name="ID2">
  <ceParams:Class>
    oracle.ds.engine.ea.compound.ServiceExecution
  </ceParams:Class>
  <ceParams:Properties>
    <ceParams:executeSingleRequest index="0">
      ServiceID1
    </ceParams:executeSingleRequest>
  </ceParams:Properties>
</ceParams:Module>
```

- executeAllRequests

With this option for the properties, the module takes the ID of the service as an element value, and executes the service with all the request messages from the request event, generating response messages and building up a response event. Example 6–34 shows a sample ServiceExecution module specification with the executeAllRequests property option.

Example 6–34 Sample Service Execution Module with the executeAllRequests Property

```
<ceParams:Module name="ID3">
  <ceParams:Class>
    oracle.ds.engine.ea.compound.ServiceExecution
  </ceParams:Class>
  <ceParams:Properties>
    <ceParams:executeAllRequests>
      serviceID1
    </ceParams:executeAllRequests>
  </ceParams:Properties>
</ceParams:Module>
```

oracle.ds.engine.ea.compound.MessageTransformer

This module transforms service messages to either requests or responses. It takes an XSLT stylesheet as a property in its properties element, and applies that XSLT stylesheet to all the incoming service messages to produce a list of outgoing services messages. Note that the MessageTransformer module can take an optional attribute index to select one message from an array of messages of a MessageSplitter module. Example 6–35 shows a sample module specification of the MessageTransformer Module. The stylesheet specified for this adaptor can use the aliases in the same way they are used by the DSXSLTInputAdaptor (see Section 6.4.1.1 for more information). If the <XSLT> element is not supplied, the MessageTransformer module behaves as if an identity transformation was applied.

The XSLTs to be applied by the MessageTransformer module must be packaged into a jar file. The location of the jar file containing the XSLTs must be specified as a service binary resource using the BINARY_RESOURCE/JAR_POINTER element in the package section of the service header. The XSLT path specified in the module properties is used to load the XSLT as a resource from that jar file.

Example 6–35 Sample MessageTransformer Module

```
<ceParams:Module name="ID4">
  <ceParams:Class>
    oracle.ds.engine.ea.compound.MessageTransformer
```

```
  </ceParams:Class>
  <ceParams:Properties>
    <ceParams:XSLT index="0">SomeXSLTURL.xsl</ceParams:XSLT>
  </ceParams:Properties>
</ceParams:Module>
```

oracle.ds.engine.ea.compound.MessageSplitter

This module splits a single message into multiple messages. It does so in one of two ways:

- SingleTransformation

  With this option of the properties, an XSLT stylesheet is specified in an element called XSLT and this stylesheet is able to transform the starting service message into a well-known structure described in Example 6–36. A list of service messages can then be generated from it.

Example 6–36 Sample Message Section of the MessageSplitter Module

```
<MESSAGES>
  <MESSAGE index="0">...</MESSAGE>
  <MESSAGE index="1">...</MESSAGE>
</MESSAGES>
```

Each message must have a valid index, and indexes must be sequential and starting from 0. If this syntax is not matched after applying the transformation to the incoming message, an exception is raised. Example 6–37 shows a sample module specification of the MessageSplitter Module using the SingleTransformation option.

Example 6–37 Sample MessageSplitter Module Using the SingleTransformation Option

```
<ceParams:Module name="ID5">
  <ceParams:Class>
    oracle.ds.engine.ea.compound.MessageSplitter
  </ceParams:Class>
  <ceParams:Properties>
    <ceParams:SingleTransformation>
      <ceParams:XSLT>SomeXSLT.xsl</ceParams:XSLT>
    </ceParams:SingleTransformation>
  </ceParams:Properties>
</ceParams:Module>
```

The stylesheet specified for this adaptor can use the aliases in the same way they are used by the DSXSLTInputAdaptor (see Section 6.4.1.1 for more information).

- MultipleTransformations

  With this option, a list of XSLT stylesheets is specified that correspond to a list of transformations performed on the original service message. This results in a list of resulting messages. With each XSLT stylesheet, an index is specified to order the list of service messages that result in the one-by-one application of the XSLT stylesheets. Example 6–38 shows a sample module specification of the MessageSplitter Module using the MultipleTransformations option.

Example 6–38 Sample MessageSplitter Module Using the Multiple Transformation Option

```
<ceParams:Module name="ID6">
  <ceParams:Class>
     oracle.ds.engine.ea.compound.MessageSplitter
  </ceParams:Class>
  <ceParams:Properties>
    <ceParams:MultipleTransformations>
      <ceParams:XSLT index="0">req2req_curr.xsl</ceParams:XSLT>
      <ceParams:XSLT index="1">req2req_pfl.xsl</ceParams:XSLT>
    </ceParams:MultipleTransformations>
  </ceParams:Properties>
</ceParams:Module>
```

The XSLTs to be applied by the MessageSplitter module must be packaged into a jar file. The location of the jar file containing the XSLTs must be specified as a service binary resource using the BINARY_RESOURCE/JAR_POINTER element in the package section of the service header. The XSLT path specified in the module properties is used to load the XSLT as a resource from that jar file. The stylesheet specified for this module can use the aliases in the same way they are used by the DSXSLTInputAdaptor (see Section 6.4.1.1 for more information). If the <XSLT> element is not supplied, the MessageSplitter module behaves as if an identity transformation was applied.

oracle.ds.engine.ea.compound.MessageMerger

This module merges multiple service messages into one single message in the form shown in Example 6–39, and then applies an XSLT stylesheet to transform that message.

The XSLT to be applied by the MessageMerger module must be packaged into a jar file. The location of the jar file containing the XSLTs must be specified as a service

binary resource using the BINARY_RESOURCE/JAR_POINTER element in the package section of the service header. The XSLT path specified in the module properties is used to load the XSLT as a resource from that jar file. The stylesheet specified for this module can use the aliases in the same way they are used by the DSXSLTInputAdaptor (see Section 6.4.1.1 for more information). If the <XSLT> element is not supplied, the MessageMerger module behaves as if an identity transformation was applied.

Example 6–39 Sample Messages Section of the MessageMerger Module

```
<MESSAGES>
  <MODULE name="ID1">
  ... msg ...
  </MODULE>
  <MODULE name="ID2">
  ... msg ...
  </MODULE>
</MESSAGES>
```

Each of the incoming messages is included in a new XML element called Module. Each of the module elements has an attribute reporting the name of the module that generated the message. Example 6–40 shows a sample module specification for the MessageMerger module.

Example 6–40 Sample MessageMerger Module

```
<ceParams:Module name="ID4">
  <ceParams:Class>
    oracle.ds.engine.ea.compound.MessageMerger
  </ceParams:Class>
  <ceParams:Properties>
    <ceParams:XSLT>resp2resp_ipfl.xsl</ceParams:XSLT>
  </ceParams:Properties>
</ceParams:Module>
```

DSCompoundServiceExecutionAdaptor acts as a coordinator to control the execution flow among the CompoundEAModules. To define the modules that participate in the execution flow, DSCompoundServiceExecutionAdaptor requires a dependency matrix parameter that is used to evaluate the execution flow. Figure 6–3 shows an example of a network of two service execution adaptors running in parallel.

Figure 6–3 Parallel Execution of Services

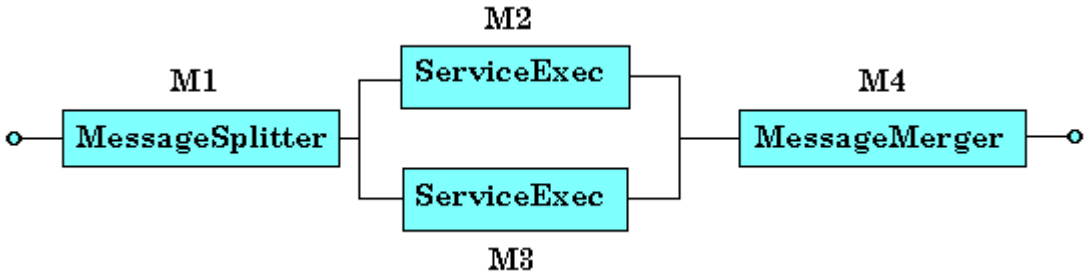


Figure 6–3 shows a possible use of the modules described previously. In this case, you can achieve the parallel execution of a couple of services by first splitting the service consumer application-supplied request, and then, joining the two services responses into one single response. The dependency matrix supplied in the DSCompoundServiceExecutionAdaptor parameters appears as shown in Example 6–41.

Example 6–41 Sample Dependency Matrix

```
<ceParams:Graph>
  <ceParams:row name="M1">
    <ceParams:column>0</ceParams:column>
    <ceParams:column>0</ceParams:column>
    <ceParams:column>0</ceParams:column>
    <ceParams:column>0</ceParams:column>
  </ceParams:row>
  <ceParams:row name="M2">
    <ceParams:column>1</ceParams:column>
    <ceParams:column>0</ceParams:column>
    <ceParams:column>0</ceParams:column>
    <ceParams:column>0</ceParams:column>
  </row>
  <ceParams:row name="M3">
    <ceParams:column>1</ceParams:column>
    <ceParams:column>0</ceParams:column>
    <ceParams:column>0</ceParams:column>
    <ceParams:column>0</ceParams:column>
  </ceParams:row>
  <ceParams:row name="M4">
    <ceParams:column>0</ceParams:column>
```

```
    <ceParams:column>1</ceParams:column>
    <ceParams:column>1</ceParams:column>
    <ceParams:column>0</ceParams:column>
  </ceParams:row>
</ceParams:Graph>
```

The matrix shows the dependencies on the input of each of the modules. DSCompoundServiceExecutionAdaptor uses this information to coordinate the execution flow, and creates the necessary threads to allow for parallel execution, if necessary. For a complete example showing how to define compound services, refer to the supplied International Portfolio (IPFL) sample service in the sample service packages directory (see Appendix D for more information).

6.4.3.3 oracle.ds.engine.ea.DSConditionalExecutionAdaptor

DSConditionalExecutionAdaptor controls the flow of execution because it executes a specified service based on the value of a certain defined alias. Service providers can configure the adaptor with switch statements that can be nested, to specify something like a decision tree, where the leaf elements are the IDs of the services to execute.

For each switch element, there is an attribute called on that you must specify to tell the adaptor which alias to switch on. In this case, only the alias is referenced, thus the value of this attribute should be just the alias name rather than {@alias}. The {@alias} value means preprocessing the value, and using the value of the alias.

For each case element under a switch element, there is an attribute called *value* that you must specify to tell the adaptor which match of the alias brings you to the inside of the case element. Inside the case element, there can be either an execute element, which means that you have reached a leaf and that its element value is the ID of the service to be executed, or you have reached another nested switch statement.

Example 6–42 shows a sample execution adaptor specification for a DSConditionalExecutionAdaptor taken from the smartlog service that comes with the installation package.

Example 6–42 Sample DSConditionalExecutionAdaptor Execution Adaptor

```
<sd:EXECUTION>
  <sd:ADAPTOR>
  <sd:NAME>oracle.ds.engine.ea.DSConditionalExecutionAdaptor</sd:NAME>
    <sd:PARAMETERS>
      <deParams:CONDITIONAL_EA_PARAMS
        xmlns:deParams="http://www.oracle.com/ds/2000/CONDITIONAL_EA_PARAMS">
```

```
        <!-- eventType has been defined previously as an alias; note, here
             we are referencing it only so the syntax used is just the alias
             name and not {@alias}. -->
        <deParams:switch on="eventType">
          <deParams:case value="CONNECT">
            <!-- Here is an example of the nesting of switch statements. -->
            <deParams:switch on="eventStatus">
              <!-- We traverse here if the eventType alias == "FAILED" -->
              <deParams:case value="FAILED">
                <!-- This is where you specify what service to execute. -->
                <deParams:execute>urn:com.oracle:ds.logger</deParams:execute>
              </deParams:case>
              <deParams:case value="CLOSE">
                <deParams:execute>urn:com.oracle:ds.logger</deParams:execute>
              </deParams:case>
              <deParams:default>urn:com.oracle:ds.logger</deParams:default>
            </deParams:switch>
          </deParams:case>

          <deParams:case value="LOOKUP">
            <deParams:switch on="eventStatus">
              <deParams:case value="OPEN">
                <deParams:execute>urn:com.oracle:ds.logger</deParams:execute>
              </deParams:case>
              <deParams:case value="FAILED">
                <deParams:execute>urn:com.oracle:ds.logger</deParams:execute>
              </deParams:case>
              <deParams:default>urn:com.oracle:ds.logger</deParams:default>
            </deParams:switch>
          </deParams:case>

        </deParams:switch>
      </deParams:CONDITIONAL_EA_PARAMS>
    </sd:PARAMETERS>
  </sd:ADAPTOR>
</sd:EXECUTION>
```

6.4.4 Output Adaptor

Section 6.4.4.1 describes the output adaptor supplied by Oracle Dynamic Services.

6.4.4.1 oracle.ds.engine.ioa.DSXSLTOutputAdaptor

DSXSLTOutputAdaptor applies an XSLT stylesheet to transform either a java.net.URLConnection or an oracle.xml.parser.v2.XMLDocument raw response. In

the case of a URLConnection, this adaptor first checks if the content-type is text/html, text/xml, or application/xml. In the HTML case, it first applies a stylesheet to transform the HTML representation into XHTML, compliant with W3C XHTML 1.0 specifications. Finally, an XSLT stylesheet is applied to the XML response. The adaptor parameters in the service descriptor define the stylesheet to apply. The stylesheet specified for this adaptor can use the aliases in the same way they are used by the DSXSLTInputAdaptor (see Section 6.4.1.1 for more information). Refer to the YahooPortfolio sample service for an example. If no <XSLT> argument is passed to the DSXSLTOutputAdaptor, the DSXSLTOutputAdaptor behaves as if an identity XSLT transformation is applied.

In the DSXSLTOutputAdaptor, there is an option of bringing in other service descriptors before applying the stylesheet. This can be done using an attribute of the xiParams:XSLT element called `applyWithServiceDescriptor`. Refer to the notifier event monitor service that comes with the Oracle Dynamic Services installation. This option can be used, for example, to bring in the logo of the service provider.

6.5 Creating Advanced Services -- Building Your Own Adaptors

Service providers can supply their own adaptors. As described previously, according to the layer or role that the adaptor addresses during the service execution, each adaptor must implement the corresponding Java interface. For example, input adaptors must implement the `oracle.ds.engine.InputAdaptor` interface and so on.

For more information about the responsibilities of each adaptor and the interfaces, see the description in the JavaDoc documentation.

For an example of building your own adaptors, refer to the YahooPortfolioCustomAdaptor demo service provided in the `ds/demo/services/YahooPortfolioCustomAdaptor` directory on Solaris or `ds\demo\services\YahooPortfolioCustomAdaptor` directory on Windows NT.

6.5.1 Packaging Your Adaptor

Once you have built your classes, grouped them into a jar file, and bundled the file into your service package, ensure the jar pointer element in the binary resource section of the service descriptor refers to the correct jar file. See Section 6.3.1.2 for more information.

7

Service Administration

In the previous chapters, some service administrator tasks included how to use the DSAdmin utility to perform basic tasks such as registering a service, creating a new service consumer application identity, and how to test a service execution. Other basic tasks such as unregistering a service, adding service consumer application properties, and so forth can also be performed using the DSAdmin utility. In this chapter, a brief overview of other topics relevant to administrators is provided.

7.1 Managing Consumer Applications

Using the DSAdmin utility, you can manage service consumer applications with respect to the application profile registry. Managing service consumer applications includes these tasks:

- Add **A:** adds the named service consumer application to the application profile registry. This operation assumes that this service consumer application name is a database user, who has been granted the connect privilege to this schema, and has been granted the DSUSER_ROLE privilege. See Section 5.1.2 for more information.
- Remove **R**: removes the named service consumer application from the application profile registry. Removing a service consumer application from the application profile registry does not remove the database user associated with it; it removes only the named service consumer application from the application profile registry.
- Grant **G**: grants the named service consumer application the privilege to execute the named service. Only a user with administrative privilege can perform this operation.
- Revoke **K**: revokes the named service consumer application the privilege to execute the named service. All properties owned by the named service

consumer application for the named service are removed. Only a user with administrative privilege can perform this operation.

- AddProp **AP**: adds a property name and property value for the named service to the named service consumer application profile. Only a user with administrative privilege can perform this operation.
- RemoveProp **RP**: removes the named property name and property value for the named service from the named service consumer application profile. Only a user with administrative privilege can perform this operation.
- GetProp **GP**: gets the named property and named property value of the named service belonging to the named service consumer application from the service consumer application profile registry.
- ListConsumers **LC**: lists all the currently registered consumer applications, the service IDs for which each service consumer application has execute privilege, and the properties of each service. Only a user with administrative privilege can perform this operation.
- ListServices **LS**: lists all service IDs for the named service consumer application to which it has been explicitly granted the privilege to access. Only a user with administrative privilege can perform this operation.
- ListProperties **LP**: lists all properties of the named service consumer application for the named service. Only a user with administrative privilege can perform this operation.

7.2 Managing Services

Using the DSAdmin utility, you can manage services with respect to the service registry. Managing services includes these tasks:

- Register **R**: registers a service in the service registry. See Section 3.2.2 for more information.
- Deregister **D**: unregisters a service from the service registry. Registered services can be unregistered using this command.
- Reregister **RR**: reregisters a service in the service registry. Previously registered services that have been unregistered, can be reregistered using this command.
- Lookup **L**: looks up a registered service in the service registry by service ID.
- Search **S**: searches the registered services in the service registry by category or keywords. See Section 3.3 for more information.

- AddCat **AC**: adds a service category to the service registry. See Section 3.2.1 for more information.
- RemoveCat **RC**: removes a service category from the service registry.

7.3 Service Response Caching

The Dynamic Services engine uses the Oracle database for caching the service responses. The caching policy for a given service is controlled through deployment parameters in the service descriptors. Before registering a service, the service administrator can review these parameters and modify them as needed. The caching parameters are defined in the SERVICE_HEADER, DEPLOYMENT, and CACHING elements in the service descriptor.

In this release, to change the caching parameters of a given service, you must unregister the service and register it again with the new parameter settings. The following information describes the caching parameters that are available:

- MAX_AGE: specifies the number of seconds the service response remains valid in the cache. After the specified amount of time elapses, the cached response is discarded. When the MAX_AGE value is specified to be zero or less, the service response is never cached.
- SESSION_PRIVATE: takes a Boolean value (TRUE or FALSE) to indicate whether cached responses for this service should be visible only within the current session, or if they should be visible to all executions. Table 7–1 shows an overview of the behavior of four possible service response cases.

Table 7–1 Possible Service Response Cases When Using a SESSION_PRIVATE Parameter Setting

Where the Service Is Executed	How the Service Response Cache Is Specified, Where the Response Is Stored, and to Whom It Is Accessible	
	The service response cache is specified as session private.	The service response cache is not specified as session private.
Service is executed within a session	The response is stored in the cache and it is accessible only to service execution within that session.	The response is stored in the cache and it is accessible to all service executions.
Service is not executed within a session	The response is not stored in the cache.	The response is stored in the cache and it is accessible to all service executions.

- USE_PROTOCOL: takes a Boolean value (TRUE or FALSE) to indicate if the expiration date of a service response should be set by what is specified through the communication protocol between the Dynamic Services engine and the remote service provider. If this parameter is true, the value of the MAX_AGE parameter described previously is ignored.

If the USE_PROTOCOL caching parameter is true, the supplied HTTP / HTTPS protocol adaptors check the Expired HTTP header to determine the expiration date of the response. The supplied JDBC protocol adaptor does not support caching.

7.4 Cache Cleanup

The cache can grow to be rather large. If caching is enabled, you may want to manually run the DS_CacheManager package procedure, deleteExpiredResponses, or start a DBMS_JOB package to periodically clean up the cache. A procedure is supplied within the DS_CacheManager package to start the DBMS_JOB package that performs the cache cleanup. This procedure is called startCleanupJob, and it takes a VARCHAR2 argument that specifies the interval between cleanup jobs.

7.5 Connecting Multiple Dynamic Services Engine Instances

Note: The information presented in this section assumes that you have performed the advanced installation option of installing the Oracle Internet Directory (OID) and set up LDAP with OID as described in Section 4.5. Once OID is installed and set up with LDAP, you can use the DSAdmin utility to manage Dynamic Services engine instances with the master registry or repository.

A logical service engine can be deployed with multiple physical service engine instances running, all sharing the same central master registry. The system can then be tuned by adding additional service execution engines. The central registry will not become a bottleneck because of the heavy use of caches at the service execution engine.

No load-balancing or failover feature is available in the current release. Administrators should partition the requests based on workload pattern. For example, an administrator can direct all applications in a subnet to a service engine in the same subnet.

There is no automatic synchronizing between multiple service engines in this current release. Administrators should synchronize all engines with the central master registry after an update. Therefore, Oracle Corporation recommends that you schedule the updates in batch mode, during low-load hours.

By default, installing a Dynamic Services engine includes:

- A registry mirror in an Oracle database
- A central master registry in a Lightweight Directory Access Protocol (LDAP) directory, for example, Oracle Internet Directory (OID)

To install an additional Dynamic Services engine:

1. Install a Dynamic Services engine as usual, but do not install the LDAP directory again.
2. Configure the LDAP connection system properties.
3. Invoke the `resync` command to resynchronize the registry. Using the DSAdmin utility, you can navigate to this command by selecting **Reg** or **R** for registry operations, and then select **Engine** or **E** to manage engine instances.

The registry should be running. Administrators should direct some users to the new Dynamic Services engines.

To perform any updates:

1. Connect to a Dynamic Services engine and perform the update (or a set of updates).
2. Connect to all other Dynamic Services engines, and invoke the `resync` command as described previously in step 3 using the DSAdmin utility.

Again, Oracle Corporation recommends that you schedule the updates in batch mode, during low-load hours.

7.6 Additional Operations of the DSAdmin Utility

You can browse through the DSAdmin utility to find additional administrative tasks that may be useful to perform.

7.6.1 Using Script Files with the DSAdmin Utility

Script files can be used with the DSAdmin utility to facilitate the process of regression testing and batch processing. Example 7–1 shows the command line and option to use is as described in Section 3.1.

Example 7–1 Run the DSAdmin Utility Using the -i Option

```
On UNIX systems:
<ORACLE_HOME>/ds/bin/dsadmin -u dssys/<dssys-password> -i <script file name>

On Windows NT systems:
<ORACLE_HOME>\ds\bin\dsadmin -u dssys/<dssys-password> -i <script file name>
```

Comments are allowed in the script in the form of lines that begin with the two slash (/ /) characters. Every string token supplied in the script file is treated as a separate command, or as user input. Commands are usually single string tokens, whereas user input need not be. For example, a category string can contain spaces within it. To use a parameter with spaces, you must enclose the entire parameter between quotation marks. This is true whether the DSAdmin utility is being used interactively or with a script.

8

Known Issues and Problems

8.1 Communications

- The current release does not throw events for warnings. So for now, the warnings take on the form of debug messages.
- If no JMS daemon is running to listen to asynchronous requests, the current implementation of the JMS driver does not time out.

8.2 Service Execution

- Service requests and responses are not validated against the corresponding XML schema.
- The FailOver adaptor does not yet enforce the service interface consistency.

8.3 Service Definitions and Creation

- Service providers defining a new service package may be able to define new adaptors.

8.4 Other Problems and Issues

For problems and issues that have become known after the release of this guide, see the online README.txt file in your *ORACLE_HOME* directory. Depending on your operating system, this file may be in:

On UNIX systems: *ORACLE_HOME*`/ds/doc/README.txt`

On Windows NT systems: *ORACLE_HOME*`\ds\doc\README.txt`

A
Links

The following is a list of Web sites that you may find useful during the use or development of services:

- W3C Extensible Markup Language (XML) 1.0 (Second Edition) Specification

  `http://www.w3.org/TR/2000/WD-xml-2e-20000814`

- W3C Extensible Stylesheet Language (XSL) Specifications

  `http://www.w3.org/TR/xsl/`

- W3C XSL Transformations (XSLT) Specifications

  `http://www.w3.org/TR/1999/PR-xslt-19991008`

- W3C XML Schema Specifications Part 0: Primer

  `http://www.w3.org/TR/xmlschema-0/`

- W3C XML Schema Specifications Part 1: Structures

  `http://www.w3.org/TR/xmlschema-1/`

- W3C XML Schema Specifications Part 2: Datatypes

  `http://www.w3.org/TR/xmlschema-2/`

- W3C Namespaces in XML

  `http://www.w3.org/TR/1999/REC-xml-names-19990114/`

- W3C Extensible HyperText Markup Language (XHTML) Specifications

  `http://www.w3.org/TR/1999/xhtml1-19990505/lastCallDiff`

- W3C HTTP 1.0 Specification

  `http://www.w3c.org/Protocols`

B

Frequently Asked Questions

A text file containing a list of frequently asked questions is available online after installing Oracle Dynamic Services.

This text file can be found at:

On UNIX systems:

$ORACLE_HOME/ds/doc/dsfaq.txt

On Windows NT systems:

$ORACLE_HOME\ds\doc\dsfaq.txt

C

Descriptive Matrix

This appendix describes the descriptive matrix of the schemas and adaptors supplied by Oracle Dynamic Services.

C.1 Syntax of the Service Descriptor Schema

At the top level, a service descriptor schema contains the data shown in Table C–1. Table Table C–2 through Table C–4 show the descriptive matrix for the classification, contact, and organization schemas. Required data is designated by **bold** element names. Element names are indented to show the relationship and hierarchy of elements. A service provider will define a service by writing an XML document that complies with the XML schema file sd.xsd and other auxiliary documents such as the classification, organization, and contact XML documents. See Example 6–11 through Example 6–15 to view the contents of sample service header and service body sections of a service descriptor schema.

Table C–1 Descriptive Matrix of the Service Descriptor Schema

Element Name	Description
SERVICE_DESCRIPTOR	Required element. The root element in the service descriptor document.
SERVICE_HEADER	Required element. The service header section. Contains high-level descriptions of the service.
NAMING	Required element. The naming section. Contains a globally unique identifier ID, as well as NAME and DESCRIPTION elements describing what the service does.
ID	Required element. An identifier uniquely identifies the service and must be a uniform resource name (URN).

Table C–1 Descriptive Matrix of the Service Descriptor Schema (Cont.)

Element Name	Description
NAME	Required element. The name of the service.
DESCRIPTION	Required element. The description of the service.
PACKAGE	Required element. The package section. Contains version specifications and pointers to where the service update is to be performed.
VERSION	Required element. The version number of the service definition package.
RELEASEDATE	Required element. The release date of the service definition package.
UPDATEURL	Optional element. The URL to obtain the latest version of the service definition package. In general, a service should contain an update URL. **Note:** However, for services created by an administrator, this parameter is meant to be used locally and UPDATEURL is not applicable (not used currently).
BINARY_RESOURCES	Optional element. The binary resources section. For advanced usage, such as specifying locations for Java class files or stylesheets for custom services and adaptors, as well as names of resource bundles.
JAR_POINTER	Optional element. The URL of the jar file containing service-specific Java classes and resources within the service definition zip file.
EXCEPTIONS	Optional element. The exceptions section. Contains the specification for the resource bundle for custom exceptions.
EXCEPTION_MSG_BUNDLE	Optional element. The specification for the custom exceptions that rely on the custom resource bundle.
DEPLOYMENT	Required element. The deployment section. Contains a set of deployment properties from the service provider to aid the service administrator during service registration.
CLASSIFICATION	Required element. An xlink to the classification XML document. Service providers can provide suggestions while a service administrator will decide the classification of the service. The XML file should comply with sd_classification.xsd. See Table C–2 for a description of the classification schema.

Table C–1 Descriptive Matrix of the Service Descriptor Schema (Cont.)

Element Name	Description
CACHING	Required element. The caching section. Contains recommended caching parameter values.
MAX_AGE	Required element. The duration that a cache entry is valid for, in *seconds*. If the value is 0, it means the entry should not be cached. The default value is 0.
SESSION_PRIVATE	Required element. A value of TRUE means that the scope of the cache entry is within the service engine user session. The default value is FALSE.
USE_PROTOCOL	Required element. A value of TRUE means that the protocol caching parameters will override MAX_AGE. The default value is FALSE.
PROVIDER	Required element. The service provider section. Contains information about the service provider including the provider's company name, copyright information, company URL, contacts for support, and URLs for logos.
ORGANIZATION	Required element. An xlink to the organization XML document. Provides generic information about the service provider. The xml file should comply with sd_organization.xsd. See Table C–4 for a description of the organization schema.
CONTACTS	Required element. The contacts section. Contains detailed support contacts for this service.
CONTACT	Required repeating element. An xlink to the contact XML document. Provides information to contact a person for any issues related to the service. The xml file should comply with sd_contact.xsd. See Table C–3 for a description of the contact schema.
INTERFACE	Required element. The service interface specification. Contains the definition of an interface characterized by the schema specifications of its input, output, and exceptions.
NAME	Required element. The name of the interface template.
INPUT_SCHEMA	Required element. An xlink to the request XML document. The URL to the request definition XML schema document, or the DTD defining the XML service request syntax.

Table C–1 Descriptive Matrix of the Service Descriptor Schema (Cont.)

Element Name	Description
OUTPUT_SCHEMA	Required element. An xlink to the response XML document. The URL to the response definition XML schema document, or the DTD defining the XML service response syntax.
SERVICE_BODY	Required element. The service body section. Contains detailed descriptions and information used by the Dynamic Services engine at execution time. Information is sectioned into specifications (including adaptors) for input, protocol, execution, and output.
INPUT	Optional element. The input section. Contains the details for preprocessing the XML request from the service consumer application and includes the following sections: namespaces, alias directives, input adaptor, and rendering directives.
NAMESPACES	Optional element. The namespaces section. Declares any namespaces and their prefixes that can be used in the aliases section to build the XPaths pointing to where the data is located.
NAMESPACE	Required repeating element. A single entry of a namespace definition.
PREFIX	Required element. The namespace prefix.
VALUE	Required element. The namespace value.
ALIASES	Optional element. The aliases section. Used by service providers to specify additional directives for the purpose of creating aliases. Aliases are used to create a map that can translate the parameters embedded in the XML document to actual parameters needed by the adaptor (for example, the protocol adaptor).
ALIAS	Required repeating element. A single entry of an alias definition.
NAME	Required element. The alias name.
VALUE	Required element. The alias value. Can be specified as an XPath or as a service consumer application profile property and optionally its modifier. The XPath is used at run-time to extract either the value of the node pointed to by the XPath, or the XML fragment (for which this node is the root) from the service request.

Table C–1 Descriptive Matrix of the Service Descriptor Schema (Cont.)

Element Name	Description
RENDERERS	Optional element. The renderers section. Contains additional rendering directives. The service provider can optionally supply some form of schema mapping specifications, such as an XSL transformation, that could map the input XML schema to a presentation form such as HTML or Wireless Markup Language (WML). Thus, the service consumer application can provide to its clients a way to enter service requests, for applications that have an HTML or WML interface.
RENDERER	Optional element. A single entry of the renderers definition.
TYPE	Optional element. The type of rendering directive, such as the request XML document or the type of input transformation, or both.
STYLESHEET	Optional element. An xlink to the request XML document or the schema mapping specification XSL transformation, or both. The URL to the request XML document or the XSL transformation mapping the input XML schema to a presentation form, or both.
ADAPTOR	Optional element. The input adaptor section. Specifies, optionally, an adaptor that further processes the service request before sending it to the service provider. A packaged adaptor is the XSLT input adaptor.
NAME	Optional element. The fully qualified name of the class implementing the oracle.ds.engine.InputAdaptor Java interface that handles the processing.
PARAMETERS	Optional element. The parameters specification. The service provider can optionally specify adaptor-specific parameters that are validated at service registration time and interpreted at runtime by the adaptor. These parameters must be in XML syntax. See Table C–5 for a description of the input adaptors parameters specification.
PROTOCOL	Optional element. The protocol section. Contains the details for submitting a request to the service provider. Identifies the way that a service engine accesses the underlying service.

Table C–1 Descriptive Matrix of the Service Descriptor Schema (Cont.)

Element Name	Description
ADAPTOR	Optional element. The protocol adaptor section. Specifies, optionally, an adaptor that transforms the standard service request into the input needed by the underlying service, using the underlying protocol.
NAME	Required element. The fully qualified name of the class implementing the oracle.ds.engine.ProtocolAdaptor interface that handles the communication to the underlying service. Packaged protocol adaptors support the HTTP, HTTPS, SMTP, or JDBC protocols.
DRIVER	Required element. The driver specification. Ensures that a certain class in the classpath for the adaptor to function properly.
PARAMETERS	Optional element. The parameters specification. The service provider can optionally specify adaptor-specific parameters that are validated at service registration time and interpreted at runtime by the adaptor. These parameters must be in XML syntax. See Table C–7, Table C–8, and Table C–9 for a description of the HTTP, JDBC, and SMTP protocol adaptors parameters specifications.
EXECUTION	Optional element. The execution section. Identifies the way in which the service must be executed. It takes the request XML and returns the response from the underlying service provider.
ADAPTOR	Optional element. The execution adaptor section. Specifies, optionally, an adaptor that executes a service request in a particular flow or order.
NAME	Optional element. The fully qualified name of the class implementing the oracle.ds.engine.ExecutionAdaptor Java interface that performs the execution. Packaged adaptors are compound, failover, and conditional service execution adaptors.
PARAMETERS	Optional element. The parameters specification. The service provider can optionally specify adaptor-specific parameters that are validated at service registration time and interpreted at runtime by the adaptor. These parameters must be in XML syntax. See Table C–10 and Table C–11 for a description of the compound and conditional execution adaptors parameters specifications.

Table C–1 Descriptive Matrix of the Service Descriptor Schema (Cont.)

Element Name	Description
OUTPUT	Optional element. The output section. Specifies the list of necessary as well as optional processing to produce the service response to the service consumer application. Includes the output adaptor and rendering directives sections.
RENDERERS	Optional element. The renderers section. Contains additional rendering directives. The service provider can optionally supply some form of schema mapping specifications, such as an XSL transformation, that could map this response XML to other forms, such as HTML or WML. Thus, the service consumer application can provide to its clients a way to produce service responses, for applications that have an HTML or WML interface.
RENDERER	Optional element. A single entry of the renderers definition.
TYPE	Optional element. The type of rendering directive, such as the response XML document or the type of output transformation, or both.
STYLESHEET	Optional element. An xlink to the response XML document or the schema mapping specification XSL transformation, or both. The URL to the response XML document or the XSL transformation mapping the output XML schema to a presentation form, or both.

Table C–1 Descriptive Matrix of the Service Descriptor Schema (Cont.)

Element Name	Description
ADAPTOR	Optional element. The output adaptor section. Specifies an output adaptor to be used to transform the output returned by the execution adaptor into an XML document compliant with the output XML schema specified in the service interface. The output name is a fully qualified name of the class implementing the oracle.ds.engine.OutputAdaptor interface that handles the transformation. A packaged adaptor is the XSLT output adaptor.
NAME	Optional element. The fully qualified name of the class implementing the oracle.ds.engine.OutputAdaptor Java interface that handles the transformation.
PARAMETERS	Optional element. The parameters specification. The service provider can optionally specify adaptor-specific parameters that are validated at service registration time and interpreted at runtime by the adaptor. These parameters must be in XML syntax. See Table C–5 for a description of the output adaptors parameters specification.

At the next level, a Classification schema contains the data shown in Table C–2. Required data is designated by **bold** element names. Element names are indented to show the relationship and hierarchy of elements. See Example 6–5 to view the contents of a classification schema.

Table C–2 Descriptive Matrix of the Classification Schema

Element Name	Description
CLASSIFICATION	Required element. The classification information from the service provider to assist the service administrator during registration.
CATEGORY	Required element. The hierarchical categories specified using the substring of the Distinguished Name (DN) specified in the RFC2253 specification (`http://www.ietf.org/`).
KEYWORDS	Required element. The keywords, separated by commas.

At the next level, a Contact schema contains the data shown in Table C–3. Required data is designated by **bold** element names. Element names are indented to show the

relationship and hierarchy of elements. See Example 6–4 to view the contents of a contact schema.

Table C–3 Descriptive Matrix of the Contact Schema

Element Name	Description
CONTACT	Required element. The contact information from the service provider.
NAME	Required element. The name of the contact.
EMAIL	Required element. The electronic mail address.
PHONE	Required element. The phone number.
FAX	Required element. The FAX number. The default is " ".
PAGER	Required element. The pager number. The default is " ".
MOBILE	Required element. The mobile number. The default is " ".

At the next level, an Organization schema contains the data shown in Table C–4. Required data is designated by **bold** element names. Element names are indented to show the relationship and hierarchy of elements. See Example 6–3 to view the contents of an organization schema.

Table C–4 Descriptive Matrix of the Organization Schema

Element Name	Description
ORGANIZATION	Required element. The company information from the service provider.
NAME	Required element. The service provider's company name.
COPYRIGHT	Required element. The copyright information for the company.
URL	Required element. The URL for the company.
LOGOURL	Required element. The URL for the company's logo.

C.2 Syntax of the Parameters Section for the Packaged Adaptors

Table C–5 through Table C–12 show the descriptive matrix for the parameters section of each type of adaptor schema supplied by Oracle Dynamic Services.

C.2.1 oracle.ds.engine.ioa.DSXSLTInputAdaptor

An Input Adaptor schema contains the data shown in Table C–5. Required data is designated by **bold** element names. Element names are indented to show the relationship and hierarchy of elements. See Example 6–13 to view the contents of the input adaptor schema contained within the service body description.

Table C–5 Descriptive Matrix of the Input Adaptor Parameters

Element Name	Description
XSLT_IA_PARAMS	Optional element. The parameters section of the input adaptor supplied by Oracle Dynamic Services.
XSLT	Optional element. The XSL stylesheet used by the input adaptor. Takes in two attributes: the isSchemaCompliant attribute specifies whether or not the XML request schema is compliant after applying the XSL stylesheet, and the applyWithServiceDescriptor attribute specifies the names of additional service descriptors to bring in before applying the stylesheet.
isSchemaCompliant	Attribute. Specifies whether or not the XML request schema is compliant after applying the XSL stylesheet. The data type is Boolean. The default value is false. This attribute is currently not in use.
applyWithServiceDescriptor	Optional attribute. Specifies the names of additional service descriptors to bring in before applying the stylesheet. The data type is string.

C.2.2 oracle.ds.engine.ioa.DSXSLTOutputAdaptor

An Output Adaptor schema contains the data shown in Table C–6. Required data is designated by **bold** element names. Element names are indented to show the relationship and hierarchy of elements. See Example 6–15 to view the contents of the output adaptor schema contained within the service body description.

Table C–6 Descriptive Matrix of the Output Adaptor Parameters

Element Name	Description
XSLT_OA_PARAMS	Optional element. The parameters section of the output adaptor supplied by Oracle Dynamic Services.
XSLT	Optional element. The XSL stylesheet used by the output adaptor. Takes in two attributes: the isSchemaCompliant attribute specifies whether or not the XML response schema is compliant after applying the XSL stylesheet, and the applyWithServiceDescriptor attribute specifies the names of additional service descriptors to bring in before applying the stylesheet.
isSchemaCompliant	Attribute. Specifies whether or not the XML response schema is compliant after applying the XSL stylesheet. The data type is Boolean. The default value is false. This attribute is currently not in use.
applyWithServiceDescriptor	Optional attribute. Specifies the names of additional service descriptors to bring in before applying the stylesheet. The data type is string.

C.2.3 oracle.ds.engine.pa.DSHTTPProtocolAdaptor

An HTTP Protocol Adaptor schema contains the data shown in Table C–7. Required data is designated by **bold** element names. Element names are indented to show the relationship and hierarchy of elements. See Example 6–27 to view the contents of the HTTP protocol adaptor parameters section contained within the service body description.

Table C–7 Descriptive Matrix of the HTTP Protocol Adaptor Parameters

Element Name	Description
HTTP_PA_PARAMS	Optional element. The section for the HTTP Protocol Adaptor parameters.
Method	Required element. The method used for the HTTP request. Must be one of three options: GET, POST, or HEAD.
URL	Required element. The URL to be contacted.
RequestHeaders	Optional element. Setting HTTP request headers in the request. Contains the definition of additional HTTP request headers that the user wants to define.

Table C–7 Descriptive Matrix of the HTTP Protocol Adaptor Parameters (Cont.)

Element Name	Description
RequestHeader	Required repeating element. The request header. Used to set the HTTP request header in the request. It is specified as a name attribute and element value of this element. Any number of request headers can be specified.
name	Attribute. The name of the request header. The data type is string.
QueryStringParameters	Optional element. The definition of the query string used for complex GET or POST requests.
QueryStringParameter	Required repeating element. The query string parameter. There can be one or more parameters for the HTTP query string. Each of these elements defines one parameter. It is specified as a "name" attribute and element value of this element. Any number of query string parameters can be specified.
name	Attribute. The name of the query string parameter. The data type is string.
Authorization	Optional element. Used for some secured Web sites that require the user's login name and password. It contains either an encoded string element or a credential element.
type	Attribute. The type of authorization. The value is fixed to the value Basic.
EncodedString	Required element. The encoded string that contains the user's login name and password. The data type is string.
Credential	Required element. The credential section. Contains the user name and password elements.
Username	Required element. The user's login name. The data type is string.
Password	Required element. The user's login password. The data type is string.

C.2.4 oracle.ds.engine.pa.DSJDBCProtocolAdaptor

A JDBC Protocol Adaptor schema contains the data shown in Table C–8. Required data is designated by **bold** element names. Element names are indented to show the relationship and hierarchy of elements. See Example 6–29 to view the contents of the JDBC protocol adaptor parameters section contained within the service body description.

Table C–8 Descriptive Matrix of the JDBC Protocol Adaptor Parameters

Element Name	Description
JDBC_PA_PARAMS	Optional element. The section for the JDBC Protocol Adaptor parameters.
Connectiondefs	Required element. The section for connection definitions.
Connection	Required repeating element. The connection section. Specifies the database connection parameters.
Username	Required element. The database user name. The data type is string.
Password	Required element. The database password. The data type is string.
dburl	Required element. The database URL. The data type is string.
driver	Optional element. The name of the Oracle JDBC driver. The data type is string.
autocommit	Required element. The automatic commit parameter value. The data type is Boolean. The default is TRUE to automatically commit any updates.
Page	Required element. The name of the connection page.
Query	Required element. The query string.

C.2.5 oracle.ds.engine.pa.DSSMTPProtocolAdaptor

An SMTP Protocol Adaptor schema contains the data shown in Table C–9. Required data is designated by **bold** element names. Element names are indented to show the relationship and hierarchy of elements. See Example 6–30 to view the contents of the SMTP protocol adaptor parameters section contained within the service body description.

Table C–9 Descriptive Matrix of the SMTP Protocol Adaptor Parameters

Element Name	Description
SMTP_PA_PARAMS	Optional element. The section for the SMTP Protocol Adaptor parameters.
Host	Required element. The SMTP host name. The data type is string.

Table C–9 Descriptive Matrix of the SMTP Protocol Adaptor Parameters (Cont.)

Port	Required element. The SMTP port number. The data type is positive integer.
From	Required element. From whom the mail was sent. The data type is string.
To	Required element. To whom the mail is to be sent. The data type is string.
cc	Optional element. To whom else should receive a copy of the mail. The data type is string.
bcc	Optional element. To whom else should receive a copy of the mail, but that is unseen by anyone else who receives a copy. The data type is string.
Subject	Required element. The subject line of the mail. The data type is string.
MsgHeaders	Optional element. The message header section.
MsgHeader	Required repeating element. The header of the mail message. It is specified as a name attribute and element value of this element.
name	Attribute. The name of the message header. The data type is string.
MsgBody	Required element. The body of the mail message.

C.2.6 oracle.ds.engine.ea.compound.DSCompoundServiceExecutionAdaptor

A Compound Execution Adaptor schema contains the data shown in Table C–10. Required data is designated by **bold** element names. Element names are indented to show the relationship and hierarchy of elements. See Example 6–32 through Example 6–41 to view the contents of the compound execution adaptor parameters sections contained within the service body description that are described in Table C–10.

Table C–10 Descriptive Matrix of the Compound Execution Adaptor Parameters

Element Name	Description
COMPOUND_EA_PARAMS	Optional element. The section for the compound execution adaptor parameters. Contains the specification that encapsulates the execution of a multitude of services, combining them into a directed graph of service executions.
Modules	Required element. The modules section.
Module	Required element. A single entry of a module. The name attribute specifies the name of the module.
Name	Attribute. The module name. The data type is string.
Class	Required element. The class type. One of four classes of compound service execution modules: *service execution*, *message transformer*, *message splitter*, and *message merger*.
Properties	Required element. Properties needed by each module.
Message Splitter Properties: choice of MultipleTransformations or SingleTransformation	
MultipleTransformations	Required element. The multiple transformations module. This module splits a single message into multiple messages, working with a specified list of XSLTs. With each XSLT, an index is also specified to order the list of service messages that result in the one-by-one application of the XSLTs. Each service message has a valid index that is sequential, starting from 0.
XSLT	Required repeating element. The specified list of XSLTs. This list corresponds to a list of transformations to the original service message, which then produces a list of resulting messages.
index	Attribute. The XSLT index number. The data type is a non-negative integer.
SingleTransformation	Required element. The single transformation module. This module splits a single message into multiple messages. The specified XSLT transforms the starting service message into a list of service messages, each with a valid index that is sequential, starting from 0.
XSLT	Required element. The specified XSLT.
Service Execution Properties: choice of executeSingleRequest or executeAllRequests	

Table C–10 Descriptive Matrix of the Compound Execution Adaptor Parameters (Cont.)

Element Name	Description
executeSingleRequest	Required element. The execute single request option. This compound service execution takes in a message index number as an attribute (a request event can contain a list of requests) and takes the ID of the service to be executed, as an element value. Only one execution is performed using one selected request message. The service specified by the ID with that request is executed.
index	Attribute. The message index number. The data type is a non-negative integer.
executeAllRequests	Required element. The execute-all-requests option. This compound service execution takes the ID of the service as an element value and executes the service with all the request messages from the request event, generating response messages to create a response event. The data type is string.
Message Merger Properties	
XSLT	Required element. The message merger XSLT. The data type is string.
Message Transformer Properties	
XSLT	Required element. The message transformer XSLT. The data type is string.
Execution Flow Definition Using a Dependency Matrix	
Graph	Required element. The dependency matrix. Contains the definition of the execution flow among a set of modules using a dependency matrix. In the dependency matrix, each module is specified by a row name and each module is also ordered by column. A dependency is represented by a column value of 1, while a value of 0 means there is no dependency.
row	Required repeating element. The row element. The name attribute specifies the row name.
name	Attribute. The name of the row. The data type is string.
column	Required repeating element. The dependency column value for each module. A value of 1 means that for the specified row, there is a dependency upon those modules representing those columns. A value of 0 means there is no dependency. The data type is a non-negative integer.

C.2.7 oracle.ds.engine.ea.DSConditionalExecutionAdaptor

A Conditional Execution Adaptor schema contains the data shown in Table C–11. Required data is designated by **bold** element names. Element names are indented to show the relationship and hierarchy of elements. See Example 6–42 to view the contents of a sample conditional execution adaptor parameters section contained within the service body description.

Table C–11 Descriptive Matrix of the Conditional Execution Adaptor Parameters

Element Name	Description
CONDITIONAL_EA_PARAMS	Optional element. The section for the conditional execution adaptor parameters. Contains an execution flow for a group of services based on a series of switch, case, and execute elements that can be nested to present a decision tree whose leaf elements are the IDs of the service to execute.
switch	Required element. The switch element. The on attribute specifies the alias name on which the switch is based.
on	Attribute. Specifies which alias to switch on. The data type is string.
case	Required repeating element. The case element. The value attribute specifies the value of the switch.
value	Attribute. Specifies which match of the alias brings you to the inside of the case element. The data type is string.
execute	Optional element. The ID of the service to execute if a case match occurs. The data type is string.
switch	Optional element. The switch element. The *on* attribute specifies the alias name on which the switch is based.
on	Attribute. Specifies which alias to switch on. The data type is string.
case	Required repeating element. The case element. The value attribute specifies the value of the switch.
value	Attribute. Specifies which match of the alias brings you to the inside of the case element. The data type is string.

Table C–11 Descriptive Matrix of the Conditional Execution Adaptor Parameters (Cont.)

execute	Optional element. The ID of the service to execute if a case match occurs. The data type is string.
default	Optional element. The branch to follow if no case match occurs.
execute	Required repeating element. The ID of the service to execute if no case match occurs. The data type is string.
default	Optional element. The branch to follow if no case match occurs.
execute	Required repeating element. The ID of the service to execute if no case match occurs. The data type is string.

C.2.8 oracle.ds.engine.ea.DSFailOverExecutionAdaptor

A Failover Execution Adaptor schema contains the data shown in Table C–12. Required data is designated by **bold** element names. Element names are indented to show the relationship and hierarchy of elements. See Example 6–31 to view the contents of a sample failover execution protocol adaptor parameters section contained within the service body description.

Table C–12 Descriptive Matrix of the Failover Execution Adaptor Parameters

Element Name	Description
FAILOVER_EA_PARAMS	Optional element. The section for the failover execution adaptor parameters. Contains an execution priority value list of services to execute in the event that a service fails to execute.
execute	Required repeating element. The execute element. The priority attribute defines the priority among the services.
priority	Attribute. The value determines the execution priority. Data type is a non-negative integer.

D

Sample Service Packages

A set of sample service packages (see Table D–1 through Table D–10) is available online after installing Oracle Dynamic Services. The sample service packages can be found at:

On UNIX systems:

$ORACLE_HOME/ds/demo/services

On Windows NT systems:

$ORACLE_HOME\ds\demo\services

These sample service packages can be installed as described in the note at the end of Section 3.2. In creating your own services, you can use these sample service packages as beginning points and copy the entire sample service package directory to a directory of your choice and then begin making modifications. Once you have determined category information for the classification XML file and have modified the remaining files to your liking, follow the steps in Section 3.2.1 to create a set of categories required by your service using the DSAdmin utility and Section 3.2.2 to register your service. Next, browse your registered services (see Section 3.3) and test your service by executing it using the DSAdmin utility (see Section 3.4).

Table D–1 CnnPortfolio Sample Service Package

Service Package	Description
CnnPortfolio	Simple service package. CNN Financial Network. Given a stock ticker, this service returns the stock ticker symbol, time, current price, change in its value, and transaction volume. This service specifies: **1.** An input section that uses no input namespaces, no aliases, specifies an input adaptor to do some validation of the request and specifically of the specified parameters, specifies rendering directives that maps the input XML schema file to HTML for presentation to service consumer application clients, then the service consumer application transforms the HTTP request back into an XML document that conforms to the request XML schema file specified by the service provider. **2.** A protocol section that uses the HTTP Web protocol to communicate to the HTTP-based service provider and specifies in the parameters section the HTTP GET request method, the servicing URL, and the HTTP query string named symbols, which resolves to the alias SymbolList. **3.** Uses the default, standard simple execution adaptor. **4.** An output section that uses an adaptor that takes the HTML response, transforms it to XHTML, then another stylesheet transforms the XHTML response to the XML response defined by the output schema; the output schema points to the XML Schema documents (or DTDs) that define the XML output the service returns to the service consumer application. The output section formats the XML response in the form of a template containing the items Symbol, Time, Price, Change, and Volume.

Table D–2 Currency Sample Service Package

Service Package	Description
Currency	Simple service package. Oanda Currency Conversion service. Given a source currency, this service returns a currency conversion value. This service specifies: **1.** An input section that uses the namespace curreq, the aliases SourceCurrency, DestCurrency, and Value, all of which specify the xpath used to extract the values of each respective node, specifies no input adaptor, specifies rendering directives that maps the input XML schema file to HTML for presentation to service consumer application clients, then the service consumer application transforms the HTTP request back into an XML document that conforms to the request XML schema file specified by the service provider. **2.** A protocol section that uses the HTTP Web protocol to communicate to the HTTP-based service provider and specifies in the parameters section the HTTP GET request method, the servicing URL, and the HTTP query strings named exch, which resolves to the alias SourceCurrency, expr, which resolves to the alias DestCurrency, and value, which resolves to the alias Value. **3.** Uses the default, standard simple execution adaptor. **4.** An output section that uses an adaptor that takes the HTML response, transforms it to XHTML, then another stylesheet transforms the XHTML response to the XML response defined by the output schema; the output schema points to the XML Schema documents (or DTDs) that define the XML output the service returns to the service consumer application. The output section formats the XML response in the form of a template containing the items SourceCurrency, DestCurrency, and Value.

Table D–3 DBService Sample Service Package

Service Package	Description
DBService	Simple service package. Database service. Given a connection name, username, password and dbURL string, this service connects to an Oracle database using the JDBC protocol adaptor, queries a table, and returns the results of the query. This service specifies: **1.** An input section that uses the namespace db, the aliases username, password, dburl, TableName, and Where, all of which specify the xpath used to extract the values of each respective node, specifies no input adaptor, specifies rendering directives that maps the input XML schema file to HTML for presentation to service consumer application clients, then the service consumer application transforms the HTTP request back into an XML document that conforms to the request XML schema file specified by the service provider. **2.** A protocol section that uses the JDBC protocol to communicate to a database-based service provider and specifies in the connectiondefs section the connection information, the connection page name and associated query string. **3.** Uses the default, standard simple execution adaptor. **4.** An output section that uses an adaptor that takes the HTML response, transforms it to XHTML, then another stylesheet transforms the XHTML response to the XML response defined by the output schema; the output schema points to the XML Schema documents (or DTDs) that define the XML output the service returns to the service consumer application. The output section formats the XML response in the form of a template containing the rowset row consisting of resultnum, employee name, and employee salary.

Table D–4 FailOverPortfolio Sample Service Package

Service Package	Description
FailOverPortfolio	Failover service package. Given a stock ticker, this service returns the current price. Should the first service not be available, a failover to a second service guarantees a result. This service specifies: **1.** An input section that uses no input namespaces, no aliases, no input adaptor, and specifies rendering directives that maps the input XML schema file to HTML for presentation to service consumer application clients, then the service consumer application transforms the HTTP request back into an XML document that conforms to the request XML schema file specified by the service provider. **2.** No protocol section. **3.** A failover execution adaptor that takes as parameters an ordered list of two compatible services. At execution time, should the first service fail to execute, the second service in the list is executed. **4.** No output section.

Table D–5 Ipfl Sample Service Package

Service Package	Description
Ipfl	Compound service package. International Portfolio service. Given a stock ticker, this service returns current prices in the supplied currency. This service specifies: **1.** An input section that uses no input namespaces, no aliases, no input adaptor, and specifies rendering directives that maps the input XML schema file to HTML for presentation to service consumer application clients, then the service consumer application transforms the HTTP request back into an XML document that conforms to the request XML schema file specified by the service provider. **2.** No protocol section. **3.** A compound execution adaptor that splits a single message into two messages, applies a list of two XSLT stylesheets by specifying an index that corresponds to the list of transformations to be performed on the original service message, which results in a list of two resulting messages. A conversion currency service and failover finance service are executed as single requests and each of the two messages is merged back into a single message. By modularizing each message, a dependency matrix is utilized that shows the dependencies of the input of each of the modules, thereby controlling the execution flow of each service on each message. The XSLTs are packaged in a jar file and its location specified as a service binary resource in the package section of the service header. **4.** No output section.

Table D–6 SampleService Sample Service Package

Service Package	Description
SampleService	Simple service package. Oanda Currency Conversion service. Note that this service contains annotations describing how its various sections work. Given a source currency, this service returns a currency conversion value. This service specifies: **1.** An input section that uses the fbreq namespace, aliases named SourceCurrency, DestCurrency, and Value, all of which specify the xpath used to extract the values of each respective node, no input adaptor, and specifies rendering directives that maps the input XML schema file to HTML for presentation to service consumer application clients, then the service consumer application transforms the HTTP request back into an XML document that conforms to the request XML schema file specified by the service provider. **2.** A protocol section that uses the HTTP Web protocol to communicate to the HTTP-based service provider and specifies in the parameters section the HTTP GET request method, the servicing URL, and the HTTP query strings named exch, which resolves to the alias SourceCurrency, expr, which resolves to the alias DestCurrency, and value, which resolves to the alias Value. **3.** Uses the default, standard simple execution adaptor. **4.** An output section that uses an adaptor that takes the HTML response, transforms it to XHTML, then another stylesheet transforms the XHTML response to the XML response defined by the output schema; the output schema points to the XML Schema documents (or DTDs) that define the XML output the service returns to the service consumer application. The output section formats the XML response in the form of a template containing the items SourceCurrency, DestCurrency, and Value.

Table D–7 Ual Sample Service Package

Service Package	Description
Ual	Simple service package. United Airlines service. Given a username and password, this service returns the mileage information from your UAL account including mileage summary, account number, membership level, and year-to-date premier miles. This service specifies: **1.** An input section that uses the namespace ualreq, the aliases AccountNumber and Password, both of which specify the xpath used to extract the values of each respective node, and specifies no input adaptor or rendering directives. **2.** A protocol section that uses the HTTP Web protocol to communicate to the HTTP-based service provider and specifies in the parameters section the HTTP POST request method, the servicing URL, the request header names to manually set the HTTP request headers in the request, and the HTTP query strings named stamp, user, which resolves to the alias AccountNumber, pwd, which resolves to the alias Password, ur_return_to, and ur_action. **3.** Uses the default, standard simple execution adaptor. **4.** An output section that uses an adaptor that takes the HTML response, transforms it to XHTML, then another stylesheet transforms the XHTML response to the XML response defined by the output schema; the output schema points to the XML Schema documents (or DTDs) that define the XML output the service returns to the service consumer application. The output section formats the XML response in the form of a template containing the items AccountNumber, MembershipLevel, PrimierMiles, and CurrentBalance.

Table D–8 Yahoo Sample Service Package

Service Package	Description
YahooPortfolio	Simple service package. Yahoo Portfolio service. Given stock tickers, this service returns the current price. This service specifies: **1.** An input section that uses no namespace, the alias SymbolList, which specifies the xpath used to extract the values of the node, specifies an input adaptor to do some validation of the request and specifically of the specified parameters, and specifies rendering directives that maps the input XML schema file to HTML for presentation to service consumer application clients, then the service consumer application transforms the HTTP request back into an XML document that conforms to the request XML schema file specified by the service provider. **2.** A protocol section that uses the HTTP Web protocol to communicate to the HTTP-based service provider and specifies in the parameters section the HTTP GET request method, the servicing URL, the HTTP query string named SymbolList, which resolves to the alias SymbolList. **3.** Uses the default, standard simple execution adaptor. **4.** An output section that uses an adaptor that takes the HTML response, transforms it to XHTML, then another stylesheet transforms the XHTML response to the XML response defined by the output schema; the output schema points to the XML Schema documents (or DTDs) that define the XML output the service returns to the service consumer application. The output section formats the XML response in the form of a template containing the items Symbol, Time, Price, Change, and Volume.

Table D–9 YahooPortfolioCustomAdaptor Sample Service Package

Service Package	Description
YahooPortfolioCustom-Adaptor	Simple service package. Yahoo Portfolio service. Given a stock ticker, this service returns the stock ticker symbol, time, current price, change in value, and transaction volume using text response. This service specifies: **1.** An input section that uses no namespace, the alias SymbolList, which specifies the xpath used to extract the values of the node, specifies an input adaptor to do some validation of the request and specifically of the specified parameters, and specifies rendering directives that maps the input XML schema file to HTML for presentation to service consumer application clients, then the service consumer application transforms the HTTP request back into an XML document that conforms to the request XML schema file specified by the service provider. **2.** A protocol section that uses the HTTP Web protocol to communicate to the HTTP-based service provider and specifies in the parameters section the HTTP GET request method, the servicing URL, the HTTP query strings named Symbols, which resolves to the alias SymbolList and format. **3.** Uses the default, standard simple execution adaptor. **4.** An output section that specifies the name of the custom output adaptor.

Table D–10 YahooPortfolioCustomProperty Sample Service Packages

Service Package	Description
YahooPortfolioCustomProperty	Simple service package. Yahoo portfolio service. Given a stock ticker, this service returns the stock ticker symbol, time, current price, change in value, and transaction volume based on the profile of the user. This service specifies: **1.** An input section that uses a ppflreq namespace, the aliases Username and SymbolList, both of which specify the xpath used to extract the values of the node, specifies no input adaptor, and specifies rendering directives that maps the input XML schema file to HTML for presentation to service consumer application clients, then the service consumer application transforms the HTTP request back into an XML document that conforms to the request XML schema file specified by the service provider. **2.** A protocol section that uses the HTTP Web protocol to communicate to the HTTP-based service provider and specifies in the parameters section the HTTP GET request method, the servicing URL, the HTTP query string named SymbolList, which resolves to the alias SymbolList. **3.** Uses the default, standard simple execution adaptor. **4.** An output section that uses an adaptor that takes the HTML response, transforms it to XHTML, then another stylesheet transforms the XHTML response to the XML response defined by the output schema; the output schema points to the XML Schema documents (or DTDs) that define the XML output the service returns to the service consumer application. The output section formats the XML response in the form of a template containing the items Symbol, Time, Price, Change, and Volume.

E

Error Messages

The following sections describe the error messages of Oracle Dynamic Services.

E.1 Execution Engine Errors

The following errors are associated with execution engine exceptions:

DS-010, Not connected

Cause: A DSConnection object attempts to make any call before the connection is established.

Action: Make sure the connection has been established for the DSConnection object.

DS-011, Already connected

Cause: A DSConnection object attempts to establish a connection after the connection is established.

Action: Make sure that the DSConnection object does not establish a connection again.

DS-020, No privilege has been granted to set DS property

Cause: A service consumer application attempts to modify Dynamic Services properties without the required privilege.

Action: Connect as DSSYS to modify a Dynamic Services property.

DS-021, No {0} property entry is found from DS property

Cause: The requested Dynamic Services property is not found.

Action: Make sure the name of the property is correct, including the case. If the name of the property is correct, contact Oracle Customer Support Services.

DS-029, Unknown error in processing DS properties

Cause: Unknown error occurred during the processing of Dynamic Services properties.

Action: Contact Oracle Customer Support Services.

DS-099, Internal Exception

Cause: An unexpected runtime exception occurred during service execution.

Action: Contact Oracle Customer Support Services.

E.2 Communication Errors

DS-101, Communication Message not valid because missing header {1}

Cause: A mandatory header of a message is missing.

Action: Contact Oracle Customer Support Services.

DS-102, Communication Message not valid because there is no XML message while being validated

Cause: The payload of a message is missing during validation.

Action: Populate the payload of the message.

DS-103, Communication Message not valid due to failure of XML parsing while being validated

Cause: The payload of a message is not well-formed XML.

Action: Correct the payload of a message so that it is well-formed.

DS-104, Communication Message not valid due to IO failed while reading from reader

Cause: Input/output related error occurred in writing the payload to a message from the designated Reader.

Action: Resolve the hardware or software failure that causes the error.

DS-105, Communication Message not valid due to IO failed while writing to writer

Cause: Input/output related error occurred in writing the payload to the designated Writer from a message.

Action: Resolve the hardware or software failure that causes the error.

DS-106, Communication Message not valid due to JMS error while reading from JMS TextMessage

Cause: Some error related to Java Messaging Service (JMS) occurred in deserializing a message from the message queue.

Action: Make sure the message queue and the associated hardware/software are functioning.

DS-107, Communication Message does not have xpath {0}

Cause: The supplied XPath, used to access a message, is not valid.

Action: Contact Oracle Customer Support Services.

DS-108, Communication Message not valid due to failure of XSLT transformation while reading message

Cause: An XSLT or XPath related error happened in processing a message.

Action: Contact Oracle Customer Support Services.

DS-109, Communication Message not valid because {0} is not of type of {1} message

Cause: The message is of unrecognized type.

Action: Contact Oracle Customer Support Services.

DS-121, Event Message not valid since it is not of type of Event message

Cause: An event message is invalid.

Action: Contact Oracle Customer Support Services.

DS-122, Event Message not valid due to JMS error while reading from JMS TextMessage

Cause: Some error related to Java Messaging Service (JMS) occurred in deserializing an event from the event message queue.

Action: Make sure the event message queue and the associated hardware/software are functioning.

DS-123, Event Message not valid due to failure of XML parsing while reading from reader

Cause: The payload of an event, generated by the service execution engine, is not well-formed XML unexpectedly.

Action: Contact Oracle Customer Support Services.

DS-132, Initialization failed due to authentication error

Cause: Invalid service consumer application credential is supplied in an attempt to connect to Dynamic Services engine.

Action: Supply a valid credential, such as a valid name and password combination. Contact your service administrator, if necessary.

DS-137, {0} failed to load due to AQ/JMS error in initializing {1}

Cause: The named module initialization failed due to failure in initializing the named queue.

Action: Make sure the installation and configuration of the named queue is correct.

DS-154, DSE Executing failed trying to decrement null Request ID

Cause: The Request ID in a response message is unexpectedly null.

Action: Contact Oracle Customer Support Services.

DS-155, DSE Execution failed with inconsistent Request Count: {0}

Cause: Request count integrity is violated in an asynchronous DSDriver.

Action: Contact Oracle Customer Support Services.

DS-156, DSE Execution failed with null request synch type

Cause: The synchronous type is unexpectedly null in a message sent through an asynchronous channel.

Action: Contact Oracle Customer Support Services.

DS-161, Publishing event failed due to IO error

Cause: An input/output related error occurred in publishing an event.

Action: Make sure the associated hardware/software are functioning.

DS162, Publishing event failed due to AQ error

Cause: An error related to the message queue occurred in publishing an event.

Action: Make sure the message queue and the associated hardware/software are functioning.

DS-181, Monitor (connect or startup) failed

Cause: Unexpected error occurred in event monitor daemon initialization.

Action: Make sure the event support installation and configuration is correct.

E.3 DS Registry Errors

DS-201, Unable to communicate with registry

Cause: The user is unable to communicate with the registry. The inability to communicate with the registry might be a result of many factors, such as network partitioning, hardware or interface problems, failures on either the client or server side.

Action: Retry the call that causes the exception. If the problem persists, contact your system administrators.

DS-206, Unable to service the request

Cause: The registry is not able to service the request. It might be unavailable for different reasons. For example, the server might be too busy to service the request, the server is running out of memory, and so forth.

Action: Retry the call that causes the exception.

DS-211, Unsupported Operation

Cause: The operation is not supported by the current release.

Action: Contact Oracle Customer Support Services for the workaround or the availability of the feature.

DS-219, Category {0} is not empty

Cause: An attempt was made to remove a service category that contains sub-categories or services.

Action: Remove all services and sub-categories under the category before re-attempting to remove the category.

DS-221, Unauthorized service consumer. login denied.

Cause: Login to the registry is denied because of improper credential.

Action: Make sure the credential is correct. If the credential is correct, contact the service administrator.

DS-222, Insufficient privileges

Cause: The service consumer application attempted to perform an administrative operation without the required privilege.

Action: Ask the service administrator to perform the administrative operation.

DS-223, Service privilege {0} is not granted to user {1}

Cause: The service consumer application does not have the service privilege

for the named service.

Action: Ask the service administrator to grant the required service privilege to the service consumer application.

DS-224, Administrative privilege is not granted to user {0}

Cause: An attempt was made to revoke the administrative privilege of the service consumer application. However, the service consumer application does not have the administrative privilege.

Action: Do not revoke the administrative privilege.

DS-231, Service {0} does not exist

Cause: An attempt was made to access a service that does not exist.

Action: Contact the service administrator to make sure that the service does exist and the required service privilege is granted.

DS-232, Category {0} does not exist

Cause: An attempt was made to access a category that does not exist.

Action: Contact the service administrator to make sure that the category does exist.

DS-233, Property {0} does not exist

Cause: An attempt was made to access a property that does not exist.

Action: Contact the service administrator to make sure that the property does exist for the connected service consumer application.

DS-234, User {0} does not exist

Cause: An attempt was made to access a service consumer application that does not exist.

Action: Contact the service administrator to make sure that the service consumer application does exist.

DS-235, Engine {0} has not been registered

Cause: An attempt was made to access the metadata of an engine instance that has not been registered.

Action: Contact the service administrator to make sure that the engine instance is registered.

DS-236, Document {0} does not exist

Cause: An attempt was made to access a document that does not exist. Exam-

ples of documents include input schema, output schema, input renderers, and output renderers.

Action: Contact the service administrator to make sure that the requested document does exist.

DS-237, Binary {0} does not exist

Cause: An attempt was made to access a binary file that does not exist. Examples include .jar file that contains specific resources or custom adaptors.

Action: Contact the service administrator to make sure that the requested binary does exist.

DS-238, Category for service {0} does not exist

Cause: During service registration, the category specified in the service descriptor does not exist in the registry.

Action: Create the required category before registering the service.

DS-251, Service {0} exists

Cause: The service to be registered already exists.

Action: If the intention is to replace the existing service, perform reregister. Otherwise, no action needs to be taken since the service already exists.

DS-252, Category {0} exists

Cause: The category to be added already exists.

Action: No action needs to be taken since the category already exists.

DS-253, Property {0} exists

Cause: The property to be added already exists.

Action: If the intention is to replace the existing property, remove the existing property before adding the property. Otherwise, no action needs to be taken since the property already exists.

DS-254, User {0} exists

Cause: The service consumer application to be added already exists.

Action: No action needs to be taken since the service consumer application already exists.

DS-255, Engine has been registered

Cause: The engine instance to be registered already exists.

Action: If the intention is to update the metadata of the engine instance, unregister the current engine instance before registering it again. Otherwise, no action needs to be taken since the engine instance has been registered.

DS-261, Invalid XML document {0}

Cause: The named XML document, typically part of a service package, is not valid.

Action: Make sure the named XML document is XML well-formed and valid.

DS-262, Invalid service package - error in accessing component {0}

Cause: An error occurred in accessing the named component of a service package. It is typically a file in a service package.

Action: Make sure the component does exist and is accessible. If the service package is a file-based service package, make sure the path is correct.

DS-263, Invalid service package - component {0} is missing its key {1}

Cause: The required element (or attribute) which serves as a key of the component is missing.

Action: Make sure the component does contain the required element (or attribute).

DS-266, Invalid or null category string {0} - syntax error

Cause: The category supplied is null or syntactically invalid.

Action: Make sure the category string is non-null and follows LDAP DN syntax.

DS-267, Invalid classification scheme constant {0}

Cause: An invalid classification scheme constant was used in conducting a search of services.

Action: Make sure the constant being used is one of those defined in the Java interface oracle.ds.registry.DSClassificationConstants.

DS-268, Invalid or null keyword string {0} - syntax error

Cause: The keyword supplied is null.

Action: Make sure the keyword string is non-null.

DS-269, Invalid or null interface string {0} - syntax error

Cause: The interface name supplied is null.

Action: Make sure the interface name string is non-null.

DS-275, Invalid engine metadata - object: {0}

Cause: The engine instance metadata supplied is null or invalid.

Action: Make sure the supplied metadata is non-null and is valid.

DS-279, Invalid object - object: {0} ; reason: {1}

Cause: The object being accessed is invalid for some unexpected reason.

Action: Contact Oracle Customer Support Services.

DS-291, Internal Error (misconfiguration) - parameter: {0}; required class type (if applicable): {1}

Cause: An unexpected mis-configuration occurred.

Action: Make sure the installation is successful. If the installation is successful and you still get this error, contact Oracle Customer Support Services.

DS-292, Internal Error (DSRegistryProvider sanity check failure) - provider class: {0}; sanity test method: {1}

Cause: An unexpected mis-configuration occurred.

Action: Contact Oracle Customer Support Services.

DS-293, Internal Error (initializing internal resources) - resource name (if known): {0}

Cause: An unexpected mis-configuration occurred.

Action: Make sure the installation is successful. If the installation is successful and you still get this error, contact Oracle Customer Support Services.

DS-294, Internal Error (unexpected null object) - object: {0}

Cause: An unexpected internal error occurred.

Action: Contact Oracle Customer Support Services.

DS-295, Internal Error (registry integrity violation) - related object: {0}

Cause: The integrity of the registry is compromised, possibly because of external intervention to the registry.

Action: Use a backup of the registry, if available. Otherwise, contact Oracle Customer Support Services.

DS-296, Internal Error (mismatch in size of write) - expected: {0} bytes; actual: {1} bytes

Cause: An unexpected internal error occurred.

Action: Contact Oracle Customer Support Services.

DS-299, Internal Error (others) - {0}

Cause: An unexpected internal error occurred.

Action: Contact Oracle Customer Support Services.

E.4 DS Engine Errors

DS-302, Invalid Request: Cannot process the request because invalid {0}

Cause: The request message sent from a service consumer application is invalid. The execution engine cannot process it.

Action: Make sure the service consumer application generates a valid request message.

DS-303, Invalid Raw Response: Cannot process the service provider raw response of type {0}

Cause: The raw response returned by the protocol adaptor is not supported by the output adaptor used.

Action: Modify and re-register the service package to use an output adaptor that supports the type of raw response returned by the protocol adaptor.

DS-304, Invalid Response: Cannot process the response of type {0}

Cause: An error occurred in processing the named response.

Action: Make sure the named response is a well-formed XML document.

DS-305, Empty Response: The service produced an empty response

Cause: The response produced is an empty one. It means that the response contains one and only one XML element and the element is an empty element without an attribute.

Action: Modify the parameters of the appropriate adaptor in the service package so that the service does not return an empty XML element only. Re-register the service package after the modification. The adaptor is typically an execution adaptor or an output adaptor.

DS-306, Invalid Session: The supplied session ID {0} is expired or invalid

Cause: The DS session ID supplied in executing a service is invalid or expired.

Action: Make sure the DS session ID is valid. Obtain a new session, if necessary.

DS-310, Invalid Adaptor: Cannot load adaptor {0}

Cause: The named adaptor cannot be loaded for any reason. A typical cause is that there is a typo in the adaptor name in the service package.

Action: Refer to the underlying exception for actions.

DS-311, Invalid Adaptor Parameter: Cannot process adaptor parameters: {0}

Cause: The adaptor parameter is invalid either during service registration or service execution.

Action: If the error occurs during service registration, check the syntax used of the adaptor parameter in the service package. If the error occurs during service execution, make sure all the other runtime requirements such as alias resolution have been satisfied.

DS-312, Invalid Alias: Required alias {0} could not be resolved

Cause: The named alias specified in the adaptor parameter is missing in the ALIASES declaration in the service package.

Action: Make sure the named alias is defined with a proper value (a proper XPath, string constant, and so forth) in the service package. Re-register the service package if any modification is made.

DS-320, Invalid XML document: {0}

Cause: The named XML document used by an adaptor is invalid.

Action: Make sure the XML document supplied is valid.

DS-321, Invalid XSL document: {0}

Cause: The named XSLT document used by an adaptor is invalid.

Action: Make sure the XSLT document supplied is valid.

DS-322, Missing Protocol Adaptor

Cause: A protocol adaptor is required in a service.

Action: Add a protocol adaptor to the service package and re-register the service package. If you believe a protocol adaptor is not required, contact the support responsible for the execution adaptor being used regarding the requirement on protocol adaptor.

DS-323, Invalid protocol {0}

Cause: The named protocol is not supported by the module.

Action: If there is an error in the name of the protocol, correct it. Otherwise, contact Oracle Customer Support Services.

DS-325, HTML Parsing errors: {0}

Cause: An error occurred in HTML parsing. This error should only occur in development tools.

Action: Make sure the HTML specified is valid.

DS-326, Element with tag {0} cannot be nested into each other, position {1}

Cause: An error occurred in HTML parsing. This error should only occur in development tools.

Action: Make sure the HTML specified is valid.

DS-327, Encountered element with tag {0} outside of a {1} tag at position {2}

Cause: An error occurred in HTML parsing. This error should only occur in development tools.

Action: Make sure the HTML specified is valid.

DS-330, Error occurred processing XSQL request: {0}

Cause: An error occurred in processing the XSQL request in DSJDBCProtocol-Adaptor.

Action: Check the adaptor parameter in the service package and make sure the XSQL request is valid.

DS-331, Error opening SQL connection. Zero or more connection definitions supplied, or invalid connection parameters: {0}

Cause: An error occurred in opening a JDBC connection in DSJDBCProtocol-Adaptor.

Action: Make sure the adaptor parameter in the service package contains connection elements with valid parameters, and the connection attribute of the page element refers to a valid connection.

DS-332, Error closing SQL connection: {0}

Cause: An error occurred in closing the JDBC connection in DSJDBCProtocol-Adaptor.

Action: Refer to the underlying SQL exception for actions.

DS-333, Error trying to open a connection with name {0}. Another connection with the same name has been found in the session, but its JDBC connection string does not match

Cause: When a service is executed in a DS session, an attempt was made to re-use an existing JDBC connection. However, the JDBC connection refers to a different database/schema.

Action: Make sure all the JDBC-based services executed in a DS session do not declare connections with the same name but different connection string.

DS-340, Error during HTTP/S connection: {0}

Cause: An HTTP transport error occurred with the named HTTP error code in DSHTTPProtocolAdaptor.

Action: Refer to the HTTP error code for resolution.

DS-341, Unsupported HTTP/S status code: {0}

Cause: The named HTTP status code is not supported by DSHTTPProtocol-Adaptor.

Action: Make sure the status code is not caused by errors in the service package or the service request. Contact Oracle Customer Support Services, if necessary.

DS-342, HTTP/S reported a fatal error with status code: {0}

Cause: An HTTP error (with status code 4XX and 5XX) was reported by the HTTP server.

Action: Refer to the underlying status code for further action. The error can also happen because of errors in the service to be executed, such as missing HTTP parameters.

DS-343, Cannot follow redirect as no Location tag has been supplied

Cause: DSHTTPProtocolAdaptor cannot follow a redirect command because the Location header is missing from the HTTP response.

Action: Contact the administrator of the HTTP server.

DS-344, Invalid HTTP/S Cookie supplied. The cookie has the wrong syntax or wrong parameters

Cause: DSHTTPProtocolAdaptor encounters an invalid cookie.

Action: Contact the administrator of the HTTP server.

DS-350, Error opening connection to SMTP host: {0}

Cause: An error occurred in establishing an SMTP connection to the named host in DSSMTPProtocolAdaptor.

Action: Make sure the host name and the port number are valid. If they are correct, contact system administrators to resolve any network, hardware, or software issues.

DS-351, Error closing connection to SMTP host: {0}

Cause: An error occurred in closing an SMTP connection to the named host in DSSMTPProtocolAdaptor.

Action: Contact the administrator of the SMTP server.

DS-352, Error sending message: {0}

Cause: An error occurred in sending the message to the SMTP server.

Action: Make sure the message is correctly formatted. If necessary, contact system administrators.

DS-353, SMTP reported fatal error with code: {0}

Cause: An SMTP error (with status code 4XX and 5XX) was reported by the SMTP server.

Action: Refer to the underlying status code for further action. The error can also happen because of errors in the service to be executed, such as missing SMTP headers, invalid email address in From header, and so forth.

DS-354, SMTP invalid mail parameter, missing {0} element

Cause: DSSMTPProtocolAdaptor reported that the named header is missing from the email to be sent.

Action: Make sure the named header is present in the adaptor parameter in the service package.

DS-360, Operation {0} on files not supported

Cause: The named operation is not supported by DSFILEProtocolAdaptor.

Action: Correct the operation to be used in the adaptor parameter in the service package.

DS-361, IOException occurred during a file operation: {0}

Cause: An Input/output related error occurred in accessing a file with the

named operation.

Action: Make sure the named operation can be done on the file.

DS-380, One or more dependent services failed during execution: {0}

Cause: One or more dependent services in a compound service, executed by DSCompoundServiceExecutionAdaptor, failed.

Action: Refer to the underlying exception for actions.

DS-383, The module ID {0} has already been defined

Cause: Multiple modules are defined with the same unique identifier incorrectly. The error is specific to DSCompoundServiceExecutionAdaptor.

Action: Find all the modules that share the same IDs in the adaptor parameter of the service package and specify different IDs for each of the modules. Re-register the modified service package.

DS-384, A module has been specified with a supplied ID or its associated class

Cause: A module is either missing the ID or the implementation class. The error is specific to DSCompoundServiceExecutionAdaptor.

Action: Fill in the missing ID or class in the Module element in the adaptor parameter of the service package. Re-register the modified service package.

DS-385, The number of rows in the execution flow matrix, {0}, does not match the number of modules defined, {1}

Cause: The Graph element in the service package, which defines the connectivity of the modules, does not map to the module declaration. The error is specific to DSCompoundServiceExecutionAdaptor.

Action: Change the Graph element or the Module elements in the adaptor parameter of the service package so that each row in the graph maps to a module. Re-register the service package.

DS-386, A row in the execution flow matrix does not specify the associated module name

Cause: The name attribute of a row element in a Graph element in the service package is missing or empty. The error is specific to DSCompoundServiceExecutionAdaptor.

Action: Find the row element and fill in the name attribute. Re-register the modified service package.

DS-387, The number of columns in the execution flow matrix, {0}, does not match the number of rows {1}

Cause: The Graph element in the service package, which defines the connectivity of the modules, does not map to the module declaration. The error is specific to DSCompoundServiceExecutionAdaptor.

Action: Change the Graph element or the Module elements in the adaptor parameter of the service package so that each column in the graph maps to a module. Re-register the modified service package.

DS-388, The execution flow matrix does not have an empty row. Such a row is necessary to identify the starting modules.

Cause: In the Graph element in the service package, there is no starting row (indicated by 0 in all column elements). Such a row is required to indicate the first module to be invoked. The error is specific to DSCompoundServiceExecutionAdaptor.

Action: Correct the Graph element so that one row is set to be a starting row: all of its column elements are 0. Re-register the modified service package.

DS-389, Module with ID {0} is not a DSEventListener

Cause: The implementation class of the named module does not implement the required DSEventListener interface. The error is specific to DSCompoundServiceExecutionAdaptor.

Action: Check the name of the implementation class of the module in the service package. Make sure the name is one of the modules defined in the User's and Administrator's Guide. Re-register the modified service package.

DS-390, Execution flow terminated with more than one message

Cause: More than one message was generated by the last module in a compound service execution flow. The error is specific to DSCompoundServiceExecutionAdaptor.

Action: Check the last module being executed and change it so that it does not generate multiple messages. For example, a ServiceExecution module should only be used with executeSingleRequest option. Re-register the modified service package.

DS-391, Module with ID {0} does not have a corresponding row in the execution flow matrix

Cause: The Graph element in the service package, which defines the connectivity of the modules, does not map to the module declaration. The error is specific to DSCompoundServiceExecutionAdaptor.

Action: Change the Graph element or the Module elements in the adaptor parameter of the service package so that each module maps to a row in the graph. Re-register the modified service package.

DS-392, Module with ID {0} could not clone a message {1}

Cause: Unexpected exception occurred in cloning a message. The error is specific to DSCompoundServiceExecutionAdaptor.

Action: Refer to the underlying exception to resolve the problem causing the error.

DS-393, The execution flow matrix contains more than one empty column. Only one empty column is necessary to identify the ending module.

Cause: In the Graph element in the service package, there are multiple modules identified as the ending module, indicated by multiple empty columns. The error is specific to DSCompoundServiceExecutionAdaptor.

Action: Correct the Graph element so that one and only one module is the ending module: one and only one column has all its values to be 0. Re-register the modified service package.

DS-395, FailOverExecution Adaptor cannot find the service to be executed with priority

Cause: An entry in the adaptor parameter of a failover service is either missing the priority or the service ID.

Action: Correct the entry so that it has both the priority and the service ID. Re-register the modified service package.

DS-396, FailOver Execution Adaptor Warning: Execution of service {0} failed ({1})

Cause: It is an indication that the named service execution fails. The failover service will attempt to execute a backup service.

Action: Service administrators should try to resolve the problem by referring to the underlying exception.

DS-397, Execution of FailOver service failed as none of the services successfully completed

Cause: A failover service execution failed because all of the backup services to be executed fail.

Action: Resolve any problems that cause the execution of the backup services to fail.

DS-398, An internal error occurred during response caching {0}

Cause: An internal error occurred in processing the Dynamic Services response cache.

Action: Refer to the underlying exception for actions.

DS-399, An internal error occurred in the execution engine {0}

Cause: An unexpected internal error occurred in service execution.

Action: Contact Oracle Customer Support Services.

E.5 DS Driver Errors

DS-401, A transport protocol error occurred for {0}. Error code: {1}

Cause: An error occurred in the transport layer between a service consumer application and a service engine. The error code, if available, is the transport-specific error code.

Action: Correct the underlying transport layer error. Contact the relevant administrator, if necessary.

DS-402, The received message has an invalid content type: {0}

Cause: The content type of the message returned by the transport layer is incorrect.

Action: Contact the service administrator.

DS-403, The received message has no transport session information

Cause: Session information is missing in the transport layer.

Action: Contact the service administrator.

E.6 DS Compound Execution Adaptor Module Errors

DS-501, Module {0} ({1}): Invalid or missing XSLT in the module parameters

Cause: The XSLT element of the named module is missing or invalid.

Action: Make sure the named module in the compound service execution adaptor parameter has a valid XSLT element.

DS-502, Module {0} ({1}): Module received {2} messages. Too many messages received

Cause: The named module receives more messages than it is designed to handle.

Action: Check the GRAPH element in the compound service execution adaptor parameter. Make sure the named module receives the correct number of messages.

DS-503, Module {0} ({1}): Module received zero or null message

Cause: The named module receives no or null message.

Action: Check the GRAPH element in the compound service execution adaptor parameter. Make sure the named module receives the correct number of messages.

DS-504, Module {0} ({1}): No message has been found at index {2}

Cause: The named module expects to receive a message with the named index out of an array of messages.

Action: Check the GRAPH element in the compound service execution adaptor parameter. Make sure the module that produces messages to the named module does create an message at the named index.

DS-505, Module {0} ({1}): XSL error applying stylesheet: {2}

Cause: The named module encounters XSLT error in applying the named XSLT stylesheet.

Action: Make sure the named XSLT stylesheet is correct with respect to the input source.

DS-508, Module {0} ({1}): Execution of service {2}, failed: {3}

Cause: The service execution of the named module fails.

Action: Check the nested exception for resolution.

DS-509, Module {0} ({1}): Internal Exception occurred: {2}

Cause: Unexpected exception occurred in the named module.

Action: Contact Oracle Customer Support Services.

DS-510, Module {0} ({1}): After applying the single transformation, the resulting message does not contain sectioning tags such as "{2}">"

Cause: The resulting XML of the named message splitter module does not contain the required sectioning tags.

Action: Check the single XSLT stylesheet used in the message splitter module and make sure the XML document generated is compliant to the specification required by single transformation in a MessageSplitter.

DS-511, Module {0} ({1}): The module {2} cannot accept more than one dependency (e.g. more than one non-null column in its row in the execution flow matrix.)

Cause: The named module cannot depend on more than one other module.

Action: Check the GRAPH element in the compound service execution adaptor parameter. Make sure the named module depends on one and only one other module.

Glossary

application profile registry

A storage place that maintains the application security profile governing service access. Registering service consumer applications allows the service administrator to choose the services that are to be visible to a particular application.

central master registry

A Lightweight Directory Access Protocol (LDAP) registry that is the main registry that can communicate with multiple Dynamic Services engine instances each containing its own registry cache. Using the DSAdmin utility, a service administrator can update the central master registry, broadcast a message to all other instances of Dynamic Services engines to manually synchronize their registry cache with the central master registry. Using a central master registry in this manner increases the scalability of Dynamic Services.

compound service package

A service package that invokes one or more other services and typically includes one additional file, a jar file, which contains all Java classes and stylesheets needed by the compound service package.

Distinguished Name (DN)

The unique name of a directory entry in Oracle Internet Directory (OID). It includes all the individual names of the parent entries back to the root. The Distinguished Name tells you exactly where the entry resides in the directory's hierarchy. This hierarchy is represented by a directory information tree (DIT).

Dynamic Services

A component of the Internet computing model that delivers a specialized value-added function. A dynamic service typically comprises some content, or some process, or both, with an open programmatic interface.

Dynamic Services engine

An engine that provides storage, access, and management of dynamic Internet and Intranet services.

Dynamic Services framework

An open, Java-based programmatic framework for enhancing Oracle as the Internet platform to incorporate, manage, and deploy dynamic Internet services. The framework includes a Dynamic Services engine, a set of dynamic services, and users of these services (service consumer applications). Oracle Dynamic Services makes it easy for application developers to rapidly incorporate existing services from a variety of Web sites, local databases, or proprietary systems into their own applications.

Dynamic Services Thin Java Client Library

Handles communication using either HTTP, HTTPS, or Java Messaging Services (JMS) communication protocol between the service consumer application and the Dynamic Services engine, which is running in the Dynamic Services gateway. Because the Dynamic Services Java client library does not contain the Dynamic Services Java engine, the Dynamic Services Java client library is referred to as a Dynamic Services thin Java client library. This is the Java (HTTP / JMS) deployment view of the Dynamic Services framework.

Dynamic Services Thick Java Client Library

Handles communication using direct Java calls between the service consumer application and the Dynamic Services engine. When the Dynamic Services Java engine is running on the machine hosting the service consumer application it is using the thick Java client library, which contains the Dynamic Services Java engine. This is the Java deployment view of the Dynamic Services framework.

execution adaptor

A routine that executes a service request in a particular flow. A flow could be as simple as relaying a request, to contacting a service provider, or as complicated as relaying a request to a service provider and relaying the response to another service provider.

input adaptor

A routine that post-processes the service input from service consumer applications to produce the standard service input that is fed to the underlying service.

monitor services

A set of services (profiler, logger, and smartlog) that are configured in the MonitorProperties.dss file for monitoring event messages generated by the Dynamic Services engine.

output adaptors

A routine that transforms the raw output from the underlying service into the standard service response.

protocol adaptor

A routine that transforms the standard service request into the input needed by the underlying service, using the underlying protocol.

service

A component within the Internet computing model that delivers a specialized value-added function.

service administrator

The person who performs administrative tasks for the Dynamic Services engine, such as enabling or disabling of services, tuning caching parameters of a service, and so forth.

service consumer application

An application that uses Dynamic Services to collect Web services from service providers and provide a dynamic service to their customers.

service descriptor

An XML schema file that defines the behavior of a service and contains service developer information, a description of service features, service management information, service input adaptors, service output adaptors, and other service provider-specific information, such as secure access, caching parameters, and so forth.

service provider

A business partner or application developer who provides and manages the content of a service for the Dynamic Services execution engine; typically, the service

provider is the owner of some data resource or process, such as, the owner of a currency exchange rate Web site. Also, someone who provides content for a service.

service registry

A storage place that maintains the service package information of registered services that enables Dynamic Services engines to set up and execute a service and access distributed sources from service providers.

simple service package

A service that is bundled into a simple service package and modeled as a local directory. This directory contains at least a `MANIFEST` file that points to the service descriptor XML file, which is the key XML document that describes the service and points to the following descriptor (.xml) and definition (.xsd) files:

- One classification XML file
- One service developer organization XML file
- One or more service developer contact XML files
- One service interface specification request (.xsd) file
- One service interface specification response (.xsd) file

Index

D

E

F

H

I

J

K

M

O

P

R

S

U

X