Oracle® Dynamic Services

User’s and Administrator’s Guide

Release 8.1.6 or higher

November 2000

Part No. Annnnn-nn

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 Dynamic Services User’s and Administrator’s Guide, Release 8.1.6 or higher

Part No. Annnnn-nn

Primary Authors: Alok Srivastava, Marco Carrier, Paul Lin, Wei Qian, Sam Lee, Kan Deng, Cheng Han

Contributing Authors: Rod Ward, Christine Chan, Katherine Oakey, Larry Guros, Yoko Mizuno

Contents

List of Examples

List of Figures

List of Tables

Send Us Your Comments

Oracle Dynamic Services User's and Administrator's Guide, Release 8.1.6

Part No. Annnnn-nn

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: 603.897.3325   Attn: Oracle Dynamic Services Documentation
- Postal service:

  Oracle Corporation
  Oracle Dynamic Services Documentation
  One Oracle Drive
  Nashua, NH 03062
  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 Oracle8*i*, Java, and XML is required.

Structure

This guide contains the following chapters and appendices:

Chapter 1	Introduces Oracle Dynamic Services; explains concepts.
Chapter 2	Describes the Oracle Dynamic Services installation and configuration.
Chapter 3	Describes how to get started in using 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.
Glossary	Describes the Dynamic Services glossary of terms.
Appendix A	Describes some helpful links to W3C specifications.
Appendix B	Describes some frequently asked questions (FAQ).

Related Documents

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

`ORACLE_HOME/ds/doc/README.txt`

Please see your operating-system specific installation guide for more information.

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

`http://technet.oracle.com/`

For more information, see the following manuals:

- *Oracle8i XML Reference*
- *PL/SQL User's Guide and Reference*
- *Oracle Internet Directory Administrator's Guide*
- Oracle8i Java Developer's Guide
- Oracle8i Java Stored Procedures Developer's Guide
- Oracle8i Enterprise JavaBeans and CORBA Developer's Guide
- Oracle8i JDBC Developer's Guide and Reference
- Oracle8i SQLJ Developer's Guide and Reference

Conventions

The following conventions are also used in this manual:

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 type 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.

1 Introduction

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

- Access sources that use a variety of protocols and APIs
- Manage sources that have different types of sessions with different protocols, that have multiple content formats, and that do not have guaranteed access
- Deliver specially formated content to both Web and mobile devices

To access dispersed sources, businesses must develop custom code to manage data transformations among different protocols, develop custom servers to use these resources, implement security, scability, and performance with every server and then manage them all. In addition, due to the extreme customization required, businesses have little or no re-usability of these custom servers.

To manage these resources, businesses must manage sessions differently for different protocols such as cookies for HTTP; they must continually rewrite URLs for database connections; they must be able to handle various content sturctures, such as Result sets, XML, Java objects, and so forth; 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 must utilize consultants to integrate with applications; they must continually change code to adapt to 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 great multitude of technical issues involved.

Application developers who develop e-Business, wireless, or portal applications must be able to easily and rapidly aggregate service offerings from service developers (business partners and application developers) and provide a single "one-stop" shopping service to customers (see Figure 1–1). Application developers, who build their service offerings upon a sound architecture, can quickly aggreagte services that are easily maintained, easily managed, and rapidly deployed to meet changing business needs. Oracle Dynamic Services is built upon a sound architecture and provides a framework for rapidly aggregating services that are truly dynamic, manageable, scalable, and secure.

As a feature of Oracle8*i*, Oracle Dynamic Services is a Java-based programmatic framework for incorporating, managing, and deploying Internet and Intranet services. Taking the Internet computing platform 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 their customers. 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 quality of service. Business opportunities can be further maximized because this framework permits customized service delivery for flexible application development.

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

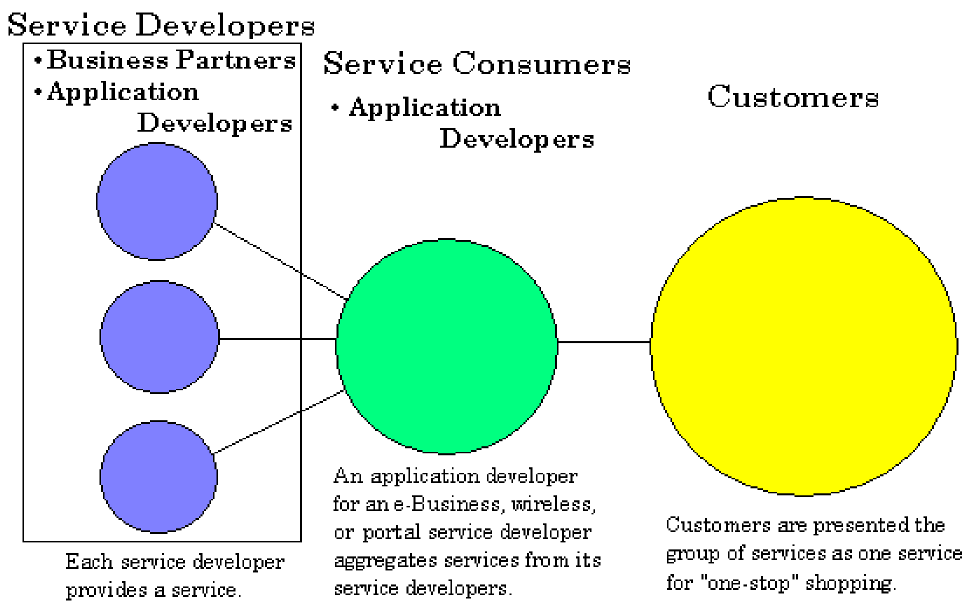
- Separation of application logic from service access for improved application development and easier maintenance.
- Repurposing available services to solve business problems, thereby reducing the cost and time needed to develop a new application.
- Making development of value-added services or applications easier and faster by brokering services from multiple resource providers.
- Enabling improvement of the used services without affecting existing applications.

Using the simple, yet flexible framework of Oracle Dynamic Services, application developers can significantly shorten the development cycle for developing applications and increase quality by selecting best-of-breed sources. With the Internet becoming the platform 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.

1.1 Application Scenarios

A service sonsumer, such as an application developer for an e-Business, wireless, or portal service developer forms relationships with its service developers, such as business partners and application developers and provides one aggregate set of services to customers as shown in Figure 1–1.

Figure 1–1 Application Developers Aggregate Services for Customers


Service Developers
• Business Partners
• Application Developers
Service Consumers
• Application Developers
Customers
Each service developer provides a service.
An application developer for an e-Business, wireless, or portal service developer aggregates services from its service developers.
Customers are presented the group of services as one service for "one-stop" shopping.

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

Wireless Service Developers

Wireless service developers can use Oracle Dynamic Services to incorporate various useful services into wireless services. Applications can be built for mobile (handheld) devices to prompt messages, advertisements, or special services based on the geographical location of the user using 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 onto a single service providing a great breadth of information to service subscribers.

Portal Service Developers

Portal service developers, who can deliver a richer, broader, 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, application developers can build applications that contain services that perform specific tasks for the user and can, for example, send them confirmation notices when each task is completed. For example, using a portal’s search engine, local restaurants of interest can be listed, a reservation made, a confirmation notice returned along with directions to the restaurant from a specific point of interest. As the success of these kinds of services grow, so does the use of the portal by its community of consumers.

e-Business Service Developers

As e-businesses grow, application developers can use Oracle Dynamic Services to integrate different related 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, then decide upon a small set of houses to visit, decide upon which house to buy, locate several mortgage lenders, fill out an online mortgage loan application, schedule home inspections, loan closures, movers, and so forth. Every possible real estate related service can be utilized 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, and so forth. Application developers for the online real estate broker can write an entire e-business application using Oracle Dynamic Services so that the real estate broker only needs to cultivate and manage the relationships among the various service providers.

1.2 Overview of Concepts

Table 1–1 describes the primary components that comprise an Oracle Dynamic Services Framework:

Table 1–1 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 and the Dynamic Services engine. Connects an application with the Dynamic Services engine. For an application to work with the Dynamic Services engine, it must first open a connection to it. This is achieved 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. The following drivers come prepackaged with Dynamic Services: ■ Direct driver -- for synchronous access to services when used in writing Internet applications ■ HTTP driver -- for remote synchronous access to services ■ JMS driver -- for remote synchronous and asynchronous access to services The communication protocol used in the connection driver implementation is completely hidden to application developers who will be always writing code using the same API. (See Section 5.1.3 for more information.)

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

Components	Functions
Service package	Contains, in its simplest form, a bundle of files modeled as a local directory comprised of a: ■ MANIFEST file that points to a service descriptor XML file that points to additional descriptor XML files: ❏ classification XML schema ❏ organization XML schema ❏ contact XML schema ❏ service request definition XML schema ❏ service response definition XML schema In addition, it specifies the service adaptors (input, protocol, execution, and output) to be used, and their parameters. Contains, in its compound form, an additional file, a jar file containing all java classes and property files 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 developers.

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

Components	**Functions**
Dynamic Services engine	Accepts service requests from client applications and does the following tasks: ▪ Determines how the service needs to be executed. ▪ Sets up the service execution environment ▪ Issues service execution requests to the resource providers. Receives the service response from the resource providers and does the following tasks: ▪ Transforms the service response for the client ▪ Returns it to the caller. The Dynamic Services engine 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 the Dynamic Services engine and all its components.

Figure 1–2 shows the Oracle Dynamic Services architecture in a conceptual way in that the Dynamic Services engine can be deployed within or outside of Oracle. Service developers (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 cached in Oracle8*i*. The Dynamic Services Java engine, depending upon the configuration, can reside either within or outside of Oracle8*i*. Dynamic Services exposes a PL/SQL interface when it runs within Oracle8*i* JVM (see Figure 1–5) or exposes a Java interface when it runs on a local machine hosting the application (thick client library) (see Figure 1–4) or 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

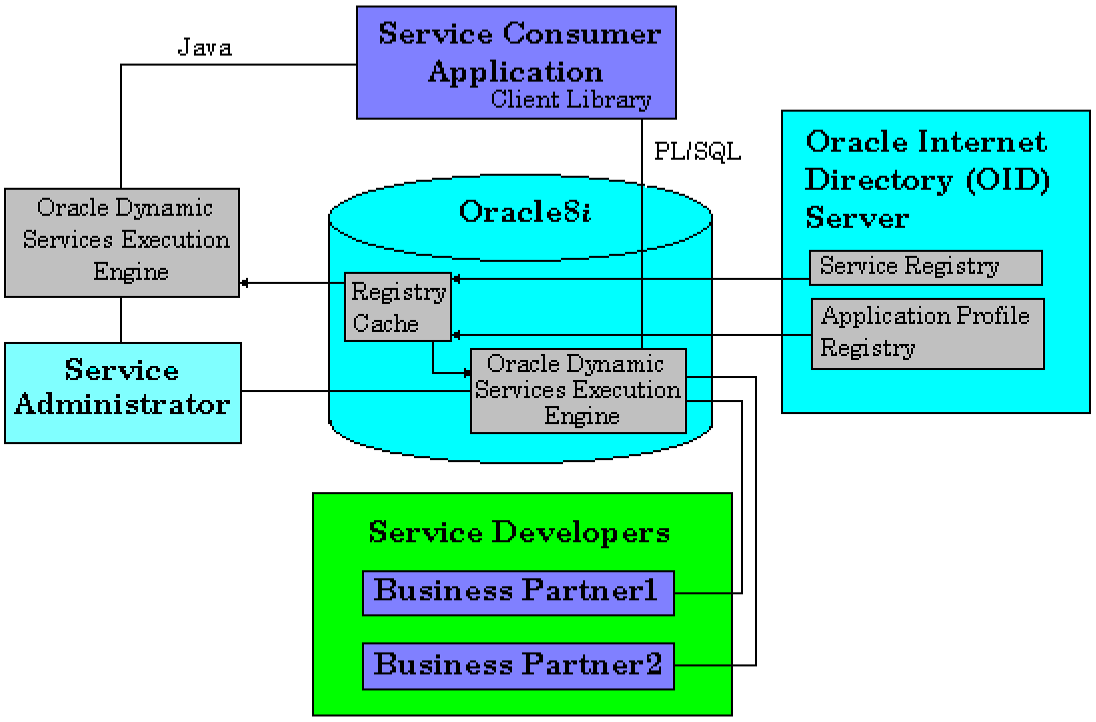

Java
Service Consumer Application
Client Library
PL/SQL
Oracle Internet Directory (OID) Server
Service Registry
Application Profile Registry
Oracle Dynamic Services Execution Engine
Oracle8i
Registry Cache
Oracle Dynamic Services Execution Engine
Service Administrator
Service Developers
Business Partner1
Business Partner2

Table 1–2 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 1–2 Summary of People or Organizations and Their Tasks or Roles in the Oracle Dynamic Services Framework

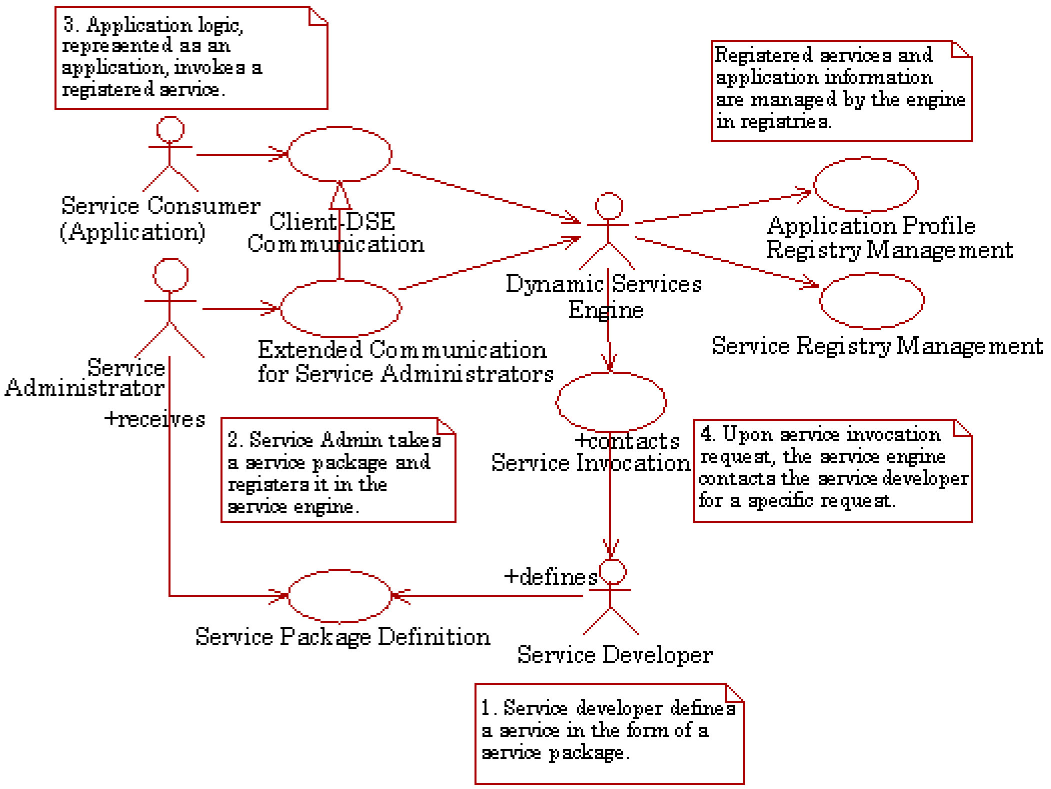
People or Organizations	Tasks or Roles
Service Provider	Provides the content for a service.
Service Developer (business partner and application developer)	Owns services Grants access to services Provides the specification for accessing services

Table 1–2 Summary of People or Organizations and Their Tasks or Roles in the Oracle Dynamic Services Framework (Cont.)

People or Organizations	Tasks or Roles
Service Developer (business partner and application developer) Service Consumer Service Administrator	Designs the service package
Service Consumer	Incorporates services into applications in order to deploy them as useful, customized services Designs the flow of the services
Service Administrator	Manages the Oracle Dynamic Services framework Manages services Grants access to services Performs tuning of the Oracle Dynamic Services engine

Figure 1–3 shows both 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 developers and service consumers 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 in the registries. 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


3. Application logic, represented as an application, invokes a registered service.
Registered services and application information are managed by the engine in registries.
Service Consumer (Application)
Client-DSE Communication
Dynamic Services Engine
Application Profile Registry Management
Service Registry Management
Service Administrator
Extended Communication for Service Administrators
+receives
2. Service Admin takes a service package and registers it in the service engine.
+contacts
Service Invocation
4. Upon service invocation request, the service engine contacts the service developer for a specific request.
+defines
Service Package Definition
Service Developer
1. Service developer defines a service in the form of a service package.

Service Provider

Service providers provide the content for a service.

Service Developer

Service developers provide and manage services that are used by the Dynamic Services engine. Service developers and service administrators define the service and load it into service registry before the service is available to applications for use. Application developer can combine many of these services to create their own value added service or application. 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 decide caching policy, failover, and so forth to further improve the Web user’s experience. The Dynamic Services engine contacts the service providers during service execution according to the specification provided in their service package that the service developer or service administrator registered.

Service Registry

The service registry is the storage place for the service package. A service package contains the information that enables Dynamic Services engines to set up and execute a service and access distributed sources from service developers. Applications can use the client library to query the Dynamic Services engine in lookup operations on the service registry. Again the actual implementation of the service registry has been abstracted out from service administrators to enable use of different registry options without affecting the client. This feature also simplifies the client applications. The registry also holds information about the identity of applications and their properties.

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 application profiles. 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 applications. The service administrator can also perform engine performance monitoring, service log reviewing, and so forth.

Application

An application acts as a client of the Dynamic Service engine. Through the Dynamic Services client API, applications acquire handles on the services it wants to execute, submits service execution requests, and collects the responses. Applications need not be aware of the communication protocol used by the Dynamic Services client library as well as the Dynamic Services engine. Such communication 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 changing business conditions and a changing technical environment that supports their applications.

Dynamic Services Engine

The core of the Dynamic Services framework is the Dynamic Services engine. The Dynamic Services engine is a multi-threaded Java engine, which accepts requests from the client applications. The Dynamic Services engine can execute services in synchronous as well as asynchronous mode 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 developers. Upon receiving the response from the service providers, the engine transforms the response for the client and returns it to the caller.

Communication Between the Application and the Dynamic Services Engine

The client library is responsible for handling the communication between the application and the Dynamic Services engine. The communication is performed as synchronous or asynchronous messages between the client library and the Dynamic Services engine. The client library is only aware of the existence of a Dynamic Services engine with which it is communicating. Applications can communicate with any available Dynamic Services engine provided they are authorized to use that particular instance. Once connected, applications have the same access and privileges as before. This feature allows distributed client access to a large number of Dynamic Services engines and can be used to implement client failovers or load balancing.

Communication Between the Service Administrator and the Dynamic Service Engine

The service administrator interfaces with the Dynamic Services engine through the administrator tools. The administrative tools use an extended version of the client library to communicate with the Dynamic Services engine. Administration can also be done through an administrative shell, shipped with the Dynamic Services engine. This is an interactive, scriptable, nested, easy-to-use shell that includes on-line help. All the features of the shell are also available from a Web-based administration utility for ease-of-use, which is shipped with the Dynamic Services engine. Service administrators can pick the utility of their choice based upon where they are working or which interface they prefer to use.

1.3 Dynamic Services Implementation Overview

Oracle Dynamic Services has three types of implementations:

- 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 behind each of the high-level components described in each implementation.

Dynamic Services Engine

The Dynamic Services engine can be deployed as a Java engine running on the machine hosting the application (thick client library) (see Figure 1–4), as a middle-tier Java engine behind a Java servlet (see Figure 1–6), or as an engine running within Oracle8*i* JVM (see Figure 1–5). Different options can be selected by users based upon their application needs. A unique feature of the Dynamic Services framework is that users can switch from one environment to another without recompiling or even restarting their applications. This gives added flexibility to the user to try out various options to see which options perform best for their applications.

Dynamic Services and Application Profile Registry

The service registry as well as 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 application. Managing services and users in the OID allows multiple instances of Dynamic Services engines to work in a synchronized fashion, giving an open, scalable option to users. For performance reasons, the registry data is cached with the Dynamic Services engine. This cache can be synchronized automatically at the start of the Dynamic Services engine, but service administrators can synchronize it through their console, or as required.

Communication Between Web Applications and the Dynamic Service Engine

The communication between the Dynamic Services engine and its applications is abstracted by the Dynamic Services client library. By registering a Dynamic Services driver, an 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). Applications have complete control over the drivers they choose within their programming framework and they can switch to any driver dynamically. Applications can use multiple drivers talking to multiple Dynamic Services engines at the same time, if their application logic requires.

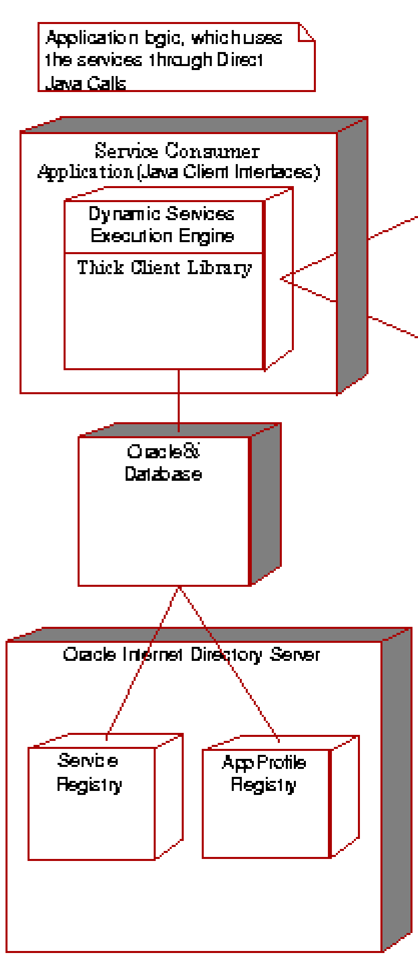
Applications can access services through different paths depending upon their Dynamic Services engine deployment. The Dynamic Services engine allows access to the services in PL/SQL and Java for programming purposes.

1.3.1 Java Deployment View

Figure 1–4 shows a basic Java deployment view of the Oracle Dynamic Services framework. The Oracle8*i* database serves as a registry cache, communicating with a Oracle Internet Directory Lightweight Directory Access Protocol (LDAP) server hosting the registries. The application contains application logic that uses the services through direct Java calls.

In this scenario, the application uses the Dynamic Services thick client library, which contains the Dynamic Services execution engine. Service developers run in their own servers.

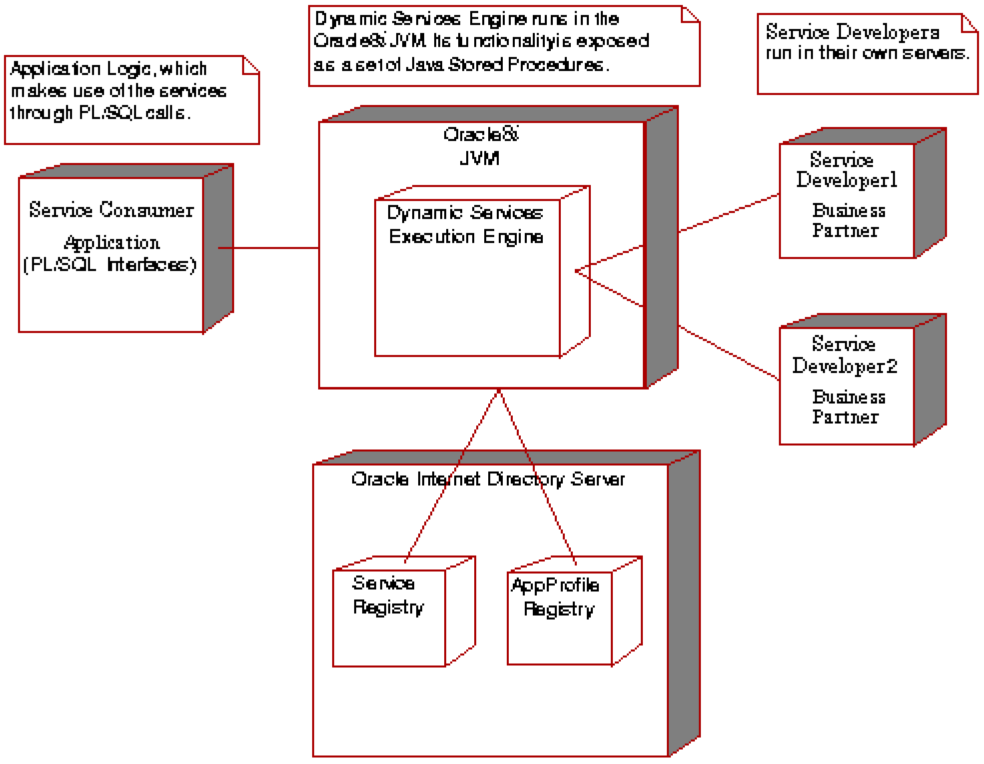
Figure 1–4 Java Deployment View of the Oracle Dynamic Services Framework


Application logic, which uses the services through Direct Java Calls
Service Developers run in their own servers.
Service Consumer Application (Java Client Interfaces)
Dynamic Services Execution Engine
Thick Client Library
Service Developer Business Partner
Service Developer Business Partner
Oracle8i Database
Oracle Internet Directory Server
Service Registry
App Profile Registry

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 Oracle8*i* JVM with its functionality exposed as a set of Java stored procedures. The Oracle8*i* database serves as a registry cache, communicating with the Oracle Internet Directory LDAP server hosting the registries. The application contains application logic, which makes use of the services through PL/SQL calls. Service developers run in their own servers.

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

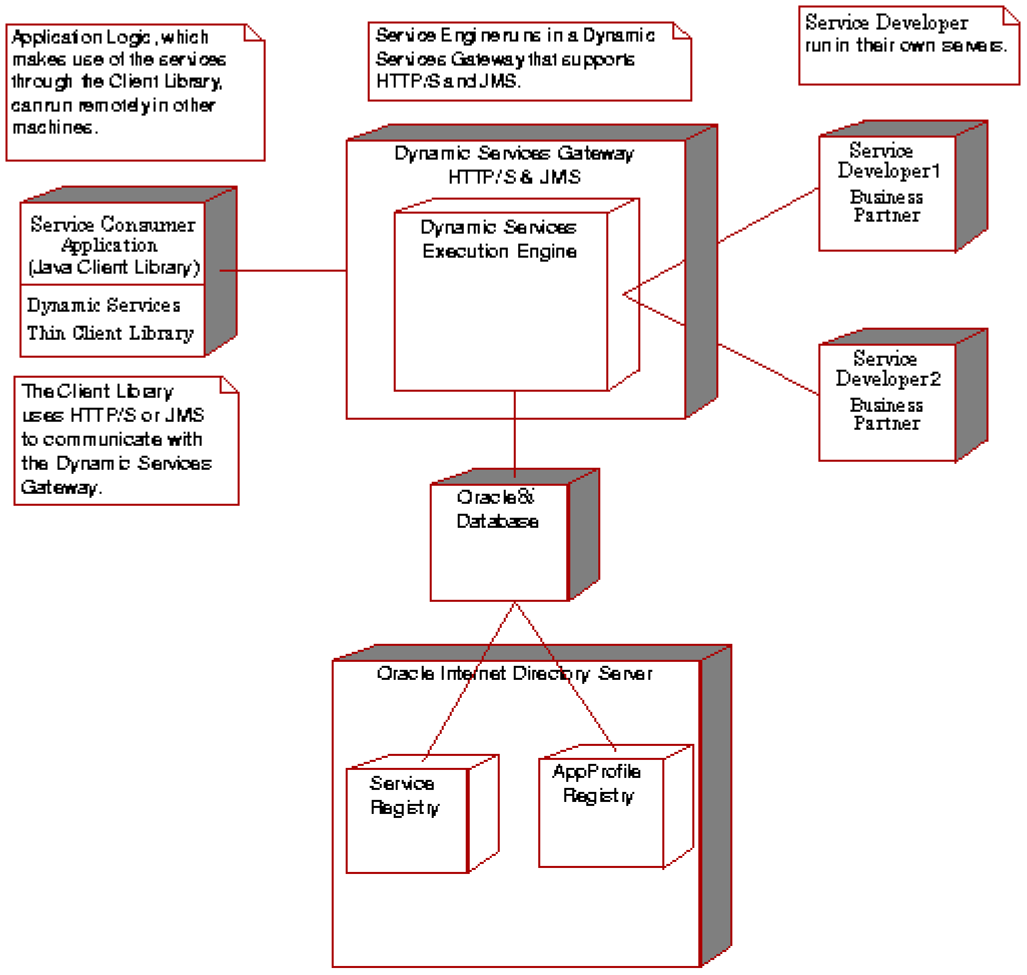

Application Logic, which makes use of the services through PL/SQL calls.
Dynamic Services Engine runs in the Oracle8i JVM. Its functionality is exposed as a set of Java Stored Procedures.
Service Developers run in their own servers.
Service Consumer Application (PL/SQL Interfaces)
Oracle8i JVM
Dynamic Services Execution Engine
Service Developer1 Business Partner
Service Developer2 Business Partner
Oracle Internet Directory Server
Service Registry
AppProfile Registry

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 that supports HTTP, HTTPS, and JMS as communication protocols. The Oracle8*i* database serves as a registry cache, communicating with the Oracle Internet Directory LDAP server hosting the registries. The 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 scenario, service execution requests are forwarded to the Dynamic Services Gateway, which executes the service and returns the response. The communication between the application and the gateway is handled by the Dynamic Services thin client library.

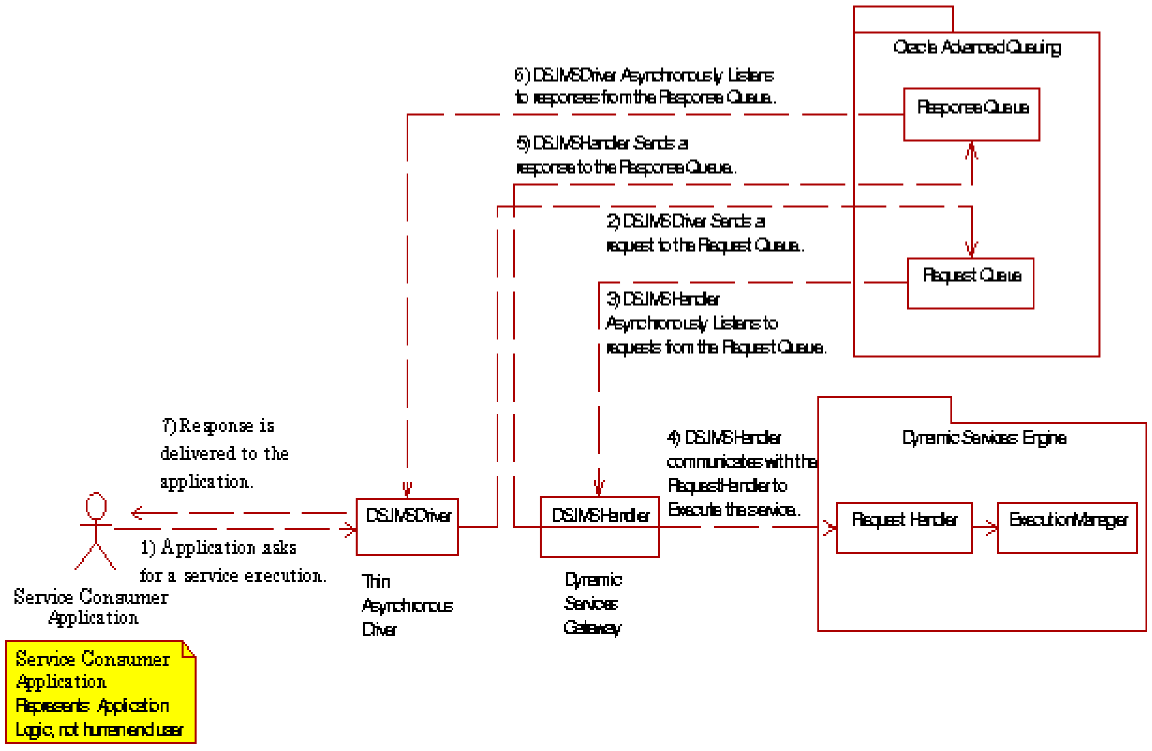
Figure 1–6 Java (HTTP/JMS) Deployment View of the Oracle Dynamic Services Framework



Application Logic, which makes use of the services through the Client Library, can run remotely in other machines.
Service Engine runs in a Dynamic Services Gateway that supports HTTP/S and JMS.
Service Developer run in their own servers.
Dynamic Services Gateway HTTP/S & JMS
Dynamic Services Execution Engine
Service Developer1 Business Partner
Service Consumer Application (Java Client Library)
Dynamic Services Thin Client Library
Service Developer2 Business Partner
The Client Library uses HTTP/S or JMS to communicate with the Dynamic Services Gateway.
Oracle8i Database
Oracle Internet Directory Server
Service Registry
AppProfile Registry

Figure 1–7 shows the asynchronous deployment communications (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. It assumes the existence of this JMS daemon running somewhere 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, generating a response, and submitting that response into another queue that the DSJMSDriver asynchronously listens to. On the application side, therefore, listeners can be registered to be informed when the response is returned.

Figure 1–7 Asynchronous Deployment Communication (JMS)


Oracle Advanced Queuing
Response Queue
Request Queue
6) DSJMSDriver Asynchronously Listens to responses from the Response Queue.
5) DSJMSHandler Sends a response to the Response Queue.
2) DSJMSDriver Sends a request to the Request Queue.
3) DSJMSHandler Asynchronously Listens to requests from the Request Queue.
4) DSJMSHandler communicates with the RequestHandler to Execute the service.
Dynamic Services Engine
Request Handler
ExecutionManager
7) Response is delivered to the application.
1) Application asks for a service execution.
DSJMSDriver
DSJMSHandler
Service Consumer Application
Thin Asynchronous Driver
Dynamic Services Gateway
Service Consumer Application Represents Application Logic, not human end user

1.4 Using Multiple Dynamic Services Engines

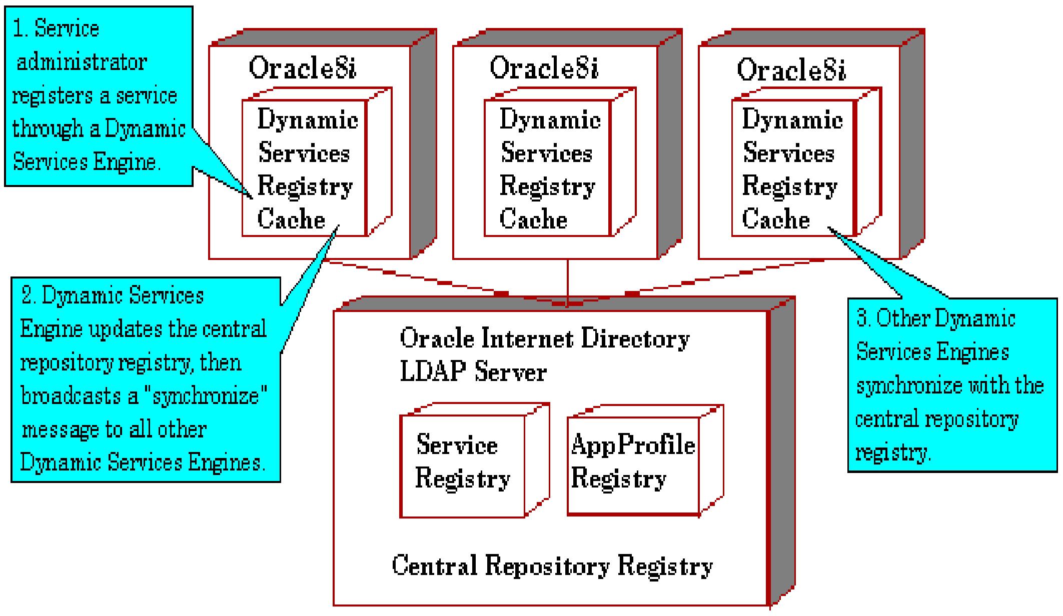
To increase scalability, you can install multiple Dynamic Services engines communicating with a central master Lightweight Directory Access Protocol

(LDAP) repository of registry information (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 master repository for registry information are as follows:

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

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


1. Service administrator registers a service through a Dynamic Services Engine.
Oracle8i
Dynamic Services Registry Cache
Oracle8i
Dynamic Services Registry Cache
Oracle8i
Dynamic Services Registry Cache
2. Dynamic Services Engine updates the central repository registry, then broadcasts a "synchronize" message to all other Dynamic Services Engines.
Oracle Internet Directory LDAP Server
Service Registry
AppProfile Registry
Central Repository Registry
3. Other Dynamic Services Engines synchronize with the central repository registry.

2 Installation and Configuration

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

Note: The version requirements for Oracle Enterprise Edition refer to releases 8.1.6, 8.1.7, and 9.0.0. If a specific release requirement is necessary, it will be noted; otherwise, the release is 8.1.6 or later.

2.1 System Requirements

The following are the system requirements:

- Oracle release: Oracle8*i* Enterprise Edition 8.1.6 or later is required to install and use Oracle Dynamic Services beta release.

Note: <*ORACLE_HOME*> is referred to as the installation directory of the Oracle8*i* 8.1.6 or later distribution.

- Hardware: The Dynamic Services engine is implemented in Java; thus, it works on any platform that has Java2-compliant (JDK 1.2.2 or later) JVM installed, including Oracle8*i* JVM or Oracle9*i* JVM.
- 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 Oracle8*i* 8.1.6 or later (a full installation in this case includes a typical Oracle8*i* 8.1.6 or later installation plus a custom installation of the JDBC Driver for Java 1.2). Otherwise, follow Oracle8*i* or later installation instructions to complete a full installation. Also, install the JDK 1.2.*n* distribution, following its installation instructions.

2.2 Dynamic Services Distribution

For release 8.1.6 or later, Oracle Dynamic Services is distributed as a zip archive file. Future distributions will be packaged with the Oracle Universal Installer. Unzip the archive file in 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.
lib	Contains the jar file of the Dynamic Services code.
classes	Contains the additional class path entry for using provider-supplied classes. This is temporary to this beta distribution and will change in future releases.
etc	Contains miscellaneous configuration files.
sql	Contains SQL scripts for installing in Oracle8*i* the support necessary for Dynamic Services.
ldif	Contains the Lightweight Directory Access Protocol (LDAP) schema definitions for the Oracle Internet Directory (OID) registry.
doc	Contains the documentation about Dynamic Services.
doc/apidocs	Contains the JavaDoc API of those classes that are necessary for service consumers and service developers to build and run services.
demo/services	Contains the sample service packages.
demo/consumer	Contains the sample client code for the service consumer.
jsp/management	Contains the Management Console application.
etc/services	Contains the service packages needed for the monitor services.
etc/xsd	Contains the XML schema for the service descriptor and supplied adaptors.
etc/dsadmin	Contains a collection of system scripts and a properties file.

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.
- 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, as shown in Example 2–1.

Example 2–1 Navigate to the Directory Where the DSSYS Schema Installation Script Is Located

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

2. Connect as SYSTEM to the Oracle8*i* 8.1.6 instance using `sqlplus` as shown in Example 2–2.

Example 2–2 Connect as SYSTEM User to the Oracle Instance

```
sqlplus system/manager
```

The DSSYS schema installation script assumes the connected 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.

3. Run the dsinstall.sql script by issuing the command in `sqlplus` as shown in Example 2–3.

Example 2–3 Run the dsinstall.sql Script

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

The dsinstall.sql script prompts you for the system password.

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 shown in Example 2–4.

Example 2–4 Run the showerrors Command to Look for Failure or Installation Errors

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

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

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

---

4. To verify the installation of the schema, exit `sqlplus` and try to reconnect to the database as the DSSYS user by issuing the command shown in Example 2–5.

Example 2–5 Verify the Installation of the DSSYS Schema by Connecting as DSSYS User

```
sqlplus dssys/dssys
```

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.

---

3 Getting Started with Dynamic Services

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.bat` on Windows NT).

Note: For Windows NT, you must first edit the `dsadmin.bat` file to specify the correct *<ORACLE_HOME>* before using the DSAdmin utility. The `dsadmin.bat` file is located in 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 the DSAdmin Utility

Before you run the DSAdmin utility, you must configure its properties file, `DSAdminShell.properties`, located in the following directory:

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

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

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

The configuration file conforms to the specifications of a Java properties file and has `name=value` pairs of properties. The specific properties that you can configure in the file are described in Table 3–1.

With the basic installation, only the DSDirectDriver driver can be used. Therefore, the only property you need to change is DS_URL_DIRECT. Change DS_URL_DIRECT to point to your database instance that hosts Oracle Dynamic Services.

Table 3–1 DSAdmin Utility Configuration Parameters

Parameter	Description
DS_DRIVERS	The value of this property is a comma-separated list of driver aliases indicating drivers that can be used to open Dynamic Services connections to be used throughout the lifetime of the shell. Typically, it does not need to be changed.
DS_DRIVER_<alias>	The name of this property is not constant and depends on an alias that is specified in DS_DRIVERS. The value of this property is the name of the actual driver class that will be loaded to set up a Dynamic Services connection. Typically, it does not need to be changed.
DS_URL_<alias>	The name of this property is not constant and depends on an alias that is specified in DS_DRIVERS. The value of this property is the name of the URL that is used by the DSDirectDriver driver to open a Dynamic Services connection. For each driver, there must be a corresponding URL, (for example, `jdbc:oracle:oci8:@db.sid` for DS_URL_Direct; `http://host-name:8888/ds/DSServlet` for DS_URL_Servlet).
DS_USERNAME	The Administrative user name to be used to log in to the Dynamic Services engine installation. The default is DSSYS.
DS_PASSWORD	The Administrative password to be used to log in to the Dynamic Services engine installation. The default is DSSYS.

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

DEF_XML_<ServiceID>	The name of this property is not constant and depends on a service ID. This property denotes 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 adding or removing the DEF_XML_<ServiceID> fields and changing the URL of the predefined driver aliases so that it points to their database instances, or the appropriate servlet URL/zones.

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

3.1.2 Running the DSAdmin Utility

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

Example 3–2 Run the DSAdmin Utility

```
<ORACLE_HOME>/ds/bin/dsadmin
```

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
-s	Executes the DSAdmin utility in silent mode. This option is used only in conjunction with a script file.

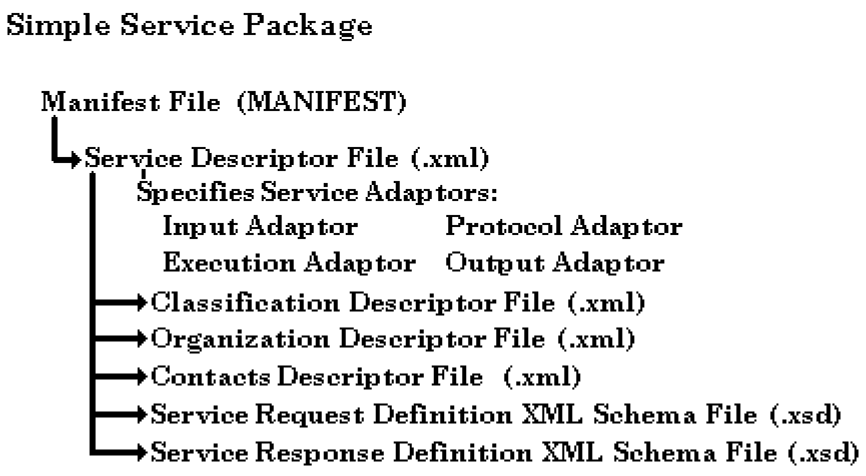
Table 3–2 DSAdmin Utility Command Line Options

-c <config file>	As mentioned previously, this option allows the DSAdmin utility to load configuration files from any location.
-i <script file>	Actions can be scriped through this option. The DSAdmin utility interprets each command separately and displays the result in standard output.

3.2 Registering a New Service

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

Figure 3–1 Contents of a Simple Service Package


Simple Service Package
Manifest File (MANIFEST)
Service Descriptor File (.xml)
Specifies Service Adaptors:
Input Adaptor
Protocol Adaptor
Execution Adaptor
Output Adaptor
Classification Descriptor File (.xml)
Organization Descriptor File (.xml)
Contacts Descriptor File (.xml)
Service Request Definition XML Schema File (.xsd)
Service Response Definition XML Schema File (.xsd)

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 XML schema file.

- Service response definition XML 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** contains multiple services and includes one additional file, a jar file, which contains all Java classes and property files needed by the compound service package.

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

The registration process involves:

1. Classifying a service under the LDAP category specified in its descriptor.
2. Registering the new service.

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).

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

```
bin/dsadmin -i demo/services/install_examples.dss
```

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:

`<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:

`/www.yahoo.com/dServices/sd/portfolio/yahoo_pfl.xml`

In the service package 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. Enter **1** or **<return>** to choose the direct driver (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

```
Register <ORACLE_HOME>/ds/demo/services/YahooPortfolio
```

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.

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**). 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. Finally, searches based on the service interface finds matches with services using the same request and response schema as those delineated by the interface.

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 3–6. The shell waits until the service execution is complete and then produces the response message.

Example 3–6 Execute a Registered Service

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

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)
- Enabling LDAP as a master repository (see Section 4.5)
- Manually fine-tuning Dynamic Services properties (see Section 4.6)
- Installing the Management Console (see Section 4.7)

Section 4.1 through Section 4.7 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.

4.1 Enabling PL/SQL Interfaces

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

A SQL script named ds_plsql.sql is provided to install the PL/SQL interface.

1. Go to the directory location (`ds/sql`) of the ds_plsql.sql file.
2. Log in to sqlplus as user DSSYS as shown in Example 4–1.

Example 4–1 Connect to the DSSYS Schema as DSSYS User

```
sqlplus DSSYS/DSSYS
```

3. Run the `ds_plsql.sql` script as shown in Example 4–2.

Example 4–2 Run the ds_plsql.sql Script

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

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

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

   Describe the Dynamic Services packages by issuing the command at a sqlplus prompt shown in Example 4–3.

Example 4–3 Describe the Dynamic Services Package Definition

```
SQL> desc DynamicServices
```

5. 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` can be found in the Dynamic Services installation directory that 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 interfaces.

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 is provided to enable the process of auditing these events. Persistent auditing is used to perform among other tasks, service execution logging and failure notification among other tasks. See Section 4.2.5 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 4–4.

Example 4–4 Set the aq_tm_process init.ora Parameter

```
aq_tm_process = 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; this is apparent in the properties of the services. If you changed the password for DSSYS, modify the default DSSYS password in this file to reflect that change.

Before installing the event monitor services, you must first configure the `MonitorProperties.dss` file in the `etc/dsadmin` directory 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 4–5.

Example 4–5 Configure the MonitorProperties.dss File

```
bin/dsadmin -i etc/dsadmin/MonitorProperties.dss
```

Then, run the `dsmoninstall.sql` script in the `ds/sql` directory.

1. Go to the directory location (`ds/sql`) of the `dsmoninstall.sql` file.
2. Log in to sqlplus as user DSSYS as shown in Example 4–6.

Example 4–6 Connect to DSSYS Schema as DSSYS User

```
sqlplus DSSYS/DSSYS
```

3. Run the dsmoninstall.sql script as shown in Example 4–7.

Example 4–7 Run the dsmoninstall.sql Script

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

A set of default services is installed from the `etc/services` directory, 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. In addition, the `dsevsp.sql` file is run to install the tables needed by these services, as most are database services. It is important to note that none of the monitor services throw 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 (`dsmon.bat` on Windows NT). 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. The correct syntax for running this utility is shown in Example 4–8.

Example 4–8 Usage Syntax for Running the Event Monitor Utility

```
dsmon dssys/dssys@dburl <oci8 | thin> <start | stop> [terse | verbose | trace]
```

Using the event monitor utility, you can choose an appropriate database URL where the event queues are hosted; 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 4–9 shows the setProperty PL/SQL procedure calls that enable event logging for the logging and warning event types.

Example 4–9 Connect to the DSSYS Schema as DSSYS User

```
connect dssys/dssys;
SQL> exec DS_Properties.setProperty('DS_EV_LOGGING_enabled', 'true');
SQL> exec DS_Properties.setProperty('DS_EV_WARNING_enabled', 'true');
```

Note: The current release does not throw events for warnings. So for now, the warnings take on the form of debug messages.

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

4.2.5 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 4–10.

Example 4–10 Definition of 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 4–11.

Example 4–11 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 4–12.

Example 4–12 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;
```

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 logging 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 implementation described in Section 1.3.3.

Dynamic Services can make use of the Apache Serlvet 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 an Apache/JServ setup. 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 `conf/jserv/` directory under your Apache installation. Configure a new `ds` mount point by adding the following new lines shown in Example 4–13.

Example 4–13 Configure a New ds Mount Point

```
# Oracle Dynamic Service 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 the following *<JAVA2_HOME>* shown in Example 4–14. *<JAVA2_HOME>* is your Java 2 SDK installation directory.

Example 4–14 Modify the wrapper.bin Line to Point to the 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 shown in Example 4–15.

Example 4–15 Create a New ds Zone

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

   c. Add a pointer to the ds zone properties as shown in Example 4–16.

Example 4–16 Add a Pointer to the ds Zone Properties

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

   d. Update Jserv Classpaths.

Add all the necessary libraries needed by Dynamic Services. Example 4–17 shows the list of necessary modifications that must be made.

Example 4–17 Update the Jserv Classpaths

```
# ----------------------------------------
# XML / XSD Parser from Oracle
# ----------------------------------------
wrapper.classpath=ORACLE_HOME/ds/lib/xmlparserv2.jar
wrapper.classpath=ORACLE_HOME/ds/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/ds/lib/providerutil.jar
wrapper.classpath=ORACLE_HOME/ds/lib/ldap.jar
wrapper.classpath=ORACLE_HOME/ds/lib/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/ds/lib/oraclexmlsql.jar
wrapper.classpath=ORACLE_HOME/ds/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 as shown in Example 4–18.

Example 4–18 Set the <ORACLE_HOME> and LD_LIBRARY_PATH Environment Variables

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

3. Edit the file *<ORACLE_HOME>*`/ds/etc/ds.properties` and make the following modifications:

   a. Update the repository location for the Dynamic Services zone

      Change the location of the Dynamic Services jar as shown in Example 4–19.

Example 4–19 Change the Location of the Dynamic Services Jar File

```
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 using the appropriate connection string for your database instance as shown in Example 4–20.

Example 4–20 Change the Driver to be Used by the DSServlet

```
servlets.default.initArgs=DS_ORCL_URL=jdbc:oracle:oci8:@ (DESCRIPTION=(ADDRESS_
LIST=(ADDRESS=(PROTOCOL=TCP)(HOST=<your-host>)(PORT=1521)))(CONNECT_DATA=(SERVICE_NAME=<your-service>)))
```

4. Restart Apache

   To restart Apache on Solaris, perform the following commands shown in Example 4–21.

Example 4–21 Restart Apache

```
cd <Apache installation directory>
bin/apachectl restart
```

4.3.2 Configuring the DSAdmin Utility to Use the HTTP Driver

After the Apache/JServ installation is complete, you can now use the DSHTTPDriver when using the DSAdmin utility after you perform the following tasks:

1. Navigate to the `etc/DSAdminShell.properties` file.

2. Enable the HTTP Driver by following the comments for the DS_DRIVERS property and update the URL used by the DSHTTPDriver by finding the line beginning with DS_URL_HTTP and 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 that 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 implementation 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`) of the dsjms_aqinit.sql file.
2. Login to sqlplus as user DSSYS as shown in Example 4–22.

Example 4–22 Connect to DSSYS Schema as DSSYS User

```
sqlplus DSSYS/DSSYS
```

3. Run the dsjms_aqinit.sql script as shown in Example 4–23.

Example 4–23 Run the dsjms_aqinit.sql Script

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

Running the dsjms_aqinit.sql script in a sqlplus 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. First edit the etc/DSJMSHandler.properties file that is used as a properties file to run the daemon. The configuration file is conformant to the specifications of

a Java properties file and contains name <space> value pairs of properties. Example 4–24 shows the specific parameters that you must configure in this properties file.

Example 4–24 Configure Parameters in the DSJMSHandler.properties File

```
# To log into the database containing the queues
USERNAME     DSSYS
PASSWORD     DSSYS

## The DB URL where the the queue and topic stay and
## where the service engine schema resides.
JMS_DB_URL   jdbc:oracle:oci8:@<your.service.name>

## The log file of the JMS handler
JMS_LOG_FILE /private/oracle/logs/jms.log

## The number of threads for the JMS handler
NUM_THREADS  10
```

> **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.

2. Run the program (`bin/dsjmsd start etc/DSJMSHandler.properties`) to start the daemon that listens to all asynchronous requests as shown in Example 4–25.

Example 4–25 Start JMS Daemon

```
bin/dsjmsd start etc/DSJMSHandler.properties
```

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 Consumer applications that want to use the JMS communication path, the database users that represent them must be granted the AQ_Administrator_Role because 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/DSAdminShell.properties` file.
2. Edit the `etc/DSAdminShell.properties` file and update the URL used by the DSJMSDriver by finding the line beginning with DS_URL_JMS and change the value to point to the URL of the database that is hosting the queues.

During run-time, requests are sent to the request queue in this database, picked up by the daemon that is communicating with this same database, and used in a service execution that returns a response that 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 Master Repository

As installed in the dsinstall.sql script, the instance of the Dynamic Services engine is a stand-alone instance with its own repository 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) repository (see Figure 1–8). First, you must successfully install the Oracle Internet Directory (OID) LDAP server with all the appropriate schemata.

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 Oracle8*i* 8.1.6 distribution and choose the Oracle8*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 shown in Example 4–26 to start it.

Example 4–26 Connect to the oidmon Instance

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

where OIDDB1 is the SID of the database instance created by the OID installer.

- Ensure the "oidldapd" server is running. If not, run the following command shown in Example 4–27 to start an instance of OID LDAP server.

Example 4–27 Run the oidctl Command to Start 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 as shown in Example 4–28.

Example 4–28 Run the ldapmodify Command to Install the Dynamic Services LDAP Schema

```
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 idapmodify command line options that can be used for installing the Dynamic Services LDAP schema:

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

Opitions	Description
h	Specifies the host machine where OID is running
p	Specifies the port number where OID is listening to. 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"
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 Oracle8*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 version of the product and the root of the User Profile Subtree. Issue the following command shown in Example 4–29 to do this.

Example 4–29 Run the ldapmodify Command to Create Default Entries for Dynamic Services

```
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, oiddsdit.ldif assumes the DN of OracleContext to be "cn=OracleContext, C=US". Change the DN to the 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 in the script ds_ldap.sql and must be configured to the appropriate server name. The two setProperty PL/SQL procedure calls are shown in Example 4–30.

Example 4–30 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 an LDAP server for a master repository of the registry rather than 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.

1. You must change `your.ldap.server` to the hostname 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 master mirror repository; however, this step is optional and needed only for management purposes.
2. Browse the `DSAdminShell.Registry.Engine` subshell to see the directives to manage the list of engines that are communicating with the master mirror repository.

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 master repository
oracle.ds.registry.ldap.principal	Username to be used to connect to LDAP server
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 Oracle8*i* database as DSSYS using SQLPlus as shown in Example 4–31.

Example 4–31 Connect to the DSSYS Schema as DSSYS User

```
sqlplus DSSYS/DSSYS
```

2. Run the setProperty PL/SQL procedure by issuing the following SQL statement as shown in Example 4–32.

Example 4–32 Run the setProperty PL/SQL Procedure

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

3. Display a listing of current properties by issuing the following SQL statements as shown in Example 4–33.

Example 4–33 Display a Listing of the Current Dynamic Services Properties

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

4.7 Installing the Management Console

The Management Console is a Java Server Page (JSP) Version 1.0 based application allowing administrators to browse and run services. Specifically, it is certified against Oracle JSP Engine release 1.0 with Apache and Apache/Jserv.

To install the management console, perform the following tasks:

1. Ensure a JSP engine is installed on the Web server to be used.
2. Copy all files in `<ORACLE_HOME>/ds/jsp/management` to a location in your Web server document path (for example, `htdocs/ds_admin`).
3. Edit `Login.jsp` file to set the default engine driver and URL as shown in Example 4–34.

Example 4–34 Edit the Login.jsp File

```
final String szDefaultUrl = "jdbc:oracle:oci8:@<db_tnsname> ";
```

4. Ensure the `lib/ds.jar` file is in the classpath of your servlet container (for example, `property wrapper.classpath in jserv.properties for Apache/JServ`), or in the classpath of the JSP engine (for example, `/WEB-INF/lib for Oracle JSP engine`).
5. Ensure <*ORACLE_HOME*> and LD_LIBRARY_PATH environment variables are properly set in your servlet container.
6. The application can be started at the following URL shown in Example 4–35.

Example 4–35 Start the Application

```
http://<your.web.server>:<your.port>/<your.path>/home.jsp
```

5 Service Consumer Interfaces

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 functionality 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 packaged in this distribution and execute them. Before proceeding, ensure that the Dynamic Services engine is properly installed and that you can register and execute services as described in the Chapter 3. Also, using the DSAdmin utility, ensure that the Yahoo portfolio service is currently 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 and to the supplied javadoc API in the `<ORACLE_HOME>/ds/doc/apidocs` directory.

5.1.1 Set the CLASSPATH

Ensure that your CLASSPATH includes all the following necessary libraries shown in Example 5–1 (that is, concatenate these paths together with a colon ":" in your CLASSPATH, (a semi-colon ";" on Windows NT)):

Example 5–1 Include these Dynamic Services Libraries in Your CLASSPATH

```
<ORACLE_HOME>/ds/lib/ds.jar
<ORACLE_HOME>/ds/lib/xmlparserv2.jar
<ORACLE_HOME>/ds/lib/xschema.jar
<ORACLE_HOME>/ds/lib/providerutil.jar
<ORACLE_HOME>/ds/lib/ldap.jar
<ORACLE_HOME>/ds/lib/jndi.jar
<ORACLE_HOME>/ds/lib/oraclexmlsql.jar
```

```
<ORACLE_HOME>/ds/lib/oraclexsql.jar
<ORACLE_HOME>/ds/lib/jsdk.jar.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 Register an Application into the Application Profile Registry

Registering an application into the Dynamic Services application profile registry is a two-step process.

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

Example 5–2 Create a New Application 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 allows the application named serviceConsumer1 to start using Dynamic Services.

Step 2: Register the user (application) identity as a new Dynamic Services Consumer (application) using the DSAdmin utility with the following commands (ensure that the user specified in the DSAdminShell.properties file is DSSYS):

- **Direct** or **1** when entering the utility, for choosing the direct driver because it is the only driver that allows for registry manipulations.
- **Reg** or **R** to enter the registry sub-shell
- **Consumer** or **C** to enter the consumer (application) management registry sub-shell
- **Add** or **A** to add a new application, followed by entering a name of a previously defined database (application) user.

  Example 5–3 shows how to add an application named serviceconsumer1 associated with the database (application) user created previously.

Example 5–3 Add an Application Associated with an Application User

```
Add serviceconsumer1
```

- **Grant** or **G** to start granting a user (application) privileges to execute services or administer the engine.
- **Service** or **1** to choose to grant service privileges, followed by the user name (application) to receive the grant and finally 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 application identified by the name, serviceconsumer1.

Example 5–4 Register the Application as a New Dynamic Services Consumer

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

In Example 5–4, urn:com.yahoo:finance.portfolio03 is the service ID of the YahooPortfolio service that is granted to the new (application) user named "serviceconsumer1" that was created in Step 1.

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

5.1.3 Open a Connection to Dynamic Services Engine

The first step that an application must perform to work with the Dynamic Services engine is to open a connection to it. This is achieved 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 engine. Applications must specify the desired driver and then operate with the returned connection. The communication protocol used in the driver implementation is completely hidden to application developers who will be always writing code using the same API. Some drivers allow for asynchronous service requests. Example 5–5 shows how to specify a driver and open a connection for an application:

Example 5–5 Specify a Driver and Open a Connection for an Application

```
// First open the connection with the Direct Driver
DSDriverManager.registerDriver("oracle.ds.driver.DSDirectDriver");
DSConnection dsconn = DSDriverManager.getConnection("jdbc:oracle:oci8:@ORCL");
dsconn.connect("ServiceConsumer1", "ServiceConsumer1");
```

5.1.3.1 Available Connection Drivers

Currently, the following drivers come prepackaged with Dynamic Services:

- *oracle.ds.driver.DSDirectDriver* for synchronous access to services when used in writing Internet applications
- *oracle.ds.driver.DSHTTPDriver* and *oracle.ds.driver.DSHTTPSDriver* for remote synchronous access to services
- *oracle.ds.driver.DSJMSDriver* for remote asynchronous access to services.

The following sections describe some important functionality differences that application developers must be aware of when using these drivers.

5.1.3.1.1 oracle.ds.driver.DSDirectDriver With the usage of the direct driver, the application assumes the Dynamic Services engine is currently available in its own CLASSPATH and therefore accessible through direct Java method calls. Through the DSConnection acquired using the direct driver, applications can perform service look-up 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 DSSYS schema is installed.

5.1.3.1.2 oracle.ds.driver.DSHTTPDriver The DSHTTPDriver allows for submission of service requests to a remote Dynamic Services engine using HTTP as a communication mechanism. The DSHTTPDriver assumes the existence of a gateway in the form of a Web server with an installed servlet that can accept service requests, 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 The 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.

Note: The implementation of this driver is based on the Oracle SSL libraries that come with Oracle 8.1.7. An Oracle 8.1.7 installation is required. Refer to the Oracle 8.1.7 SSL instructions.

5.1.3.1.4 oracle.ds.driver.DSJMSDriver 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. It assumes the existence of this JMS daemon running somewhere 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, generating a response, and submitting that response into another queue that the DSJMSDriver asynchronously listens to. On the 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 two drivers requires no modifications in the application code other than the registration of the driver itself.

5.1.4 Example: Executing the Yahoo Portfolio Service

The steps required to execute a 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 Example Request of 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 request XML document.

Example 5–7 describes the example request of the Yahoo portfolio service in the pfl_req_ex.xml file in the ds/demo/services/YahooPortfolio directory.

Example 5–7 Example Request of the Yahoo Portfolio Service

```
<?xml version="1.0"?>
<!-- Example request of the Yahoo! portfolio 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 be compliant with request syntax defined for the Yahoo service.

5.1.5 Display Service Response

Once a service response has been obtained, its content can be parsed by the Oracle XML parser and printed as hown 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 Application Sessions

The life-cycle of a Dynamic Services application requires it to open a connection to the Dynamic Services engine at the beginning. Within this connection, applications can execute multiple services. Each of these services can actually create a session with the remote service developer. 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 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, applications obtain an "opaque" session identifier. To continue the session, they must set the session identifier in the header of those service requests they want to execute within the session. Corresponding DSResponses each contain header information about the session they belong to. To close a session, applications can use the DSConnection.closeSession() method, that 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 only persists through the life-cycle of the host JVM where the Dynamic Services engine is running.

5.2 PL/SQL Interface for Service Consumers

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 DSSYS schema explicitly must have the following additional GRANT statements shown in Example 5–9.

Example 5–9 Use these Grant Statements to Access the PL/SQL Dynamic Services 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 an application that uses Dynamic services, you can inspect the `createPLSQLConsumer.sql` script file under the `demo/consumer` directory. For more details about users and the database, refer to Oracle8*i* documentation.

As described in Section 1.3.2, in a PL/SQL deployment of Dynamic Services, the Dynamic Services engine runs in the Oracle8*i* JVM and its functionality is exposed as a set of Java stored procedures through a PL/SQL interface (see Example 5–10); note the PL/SQL procedures and functions that are defined for this interface. An application using this PL/SQL interface contains application logic that makes use of these services through PL/SQL calls to these procedures and functions as shown in Example 5–11.

Example 5–10 PL/SQL Interface for Dynamic Services

```
-- This procedure initializes the Dynamic Services engine within the 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
PROCEDURE close;

-- This function executes a service given a 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
  FUNCTION execute(service_id VARCHAR2, request VARCHAR2) RETURN VARCHAR2;

  -- This executes a service given a 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 traserve down and an entry
  -- in the supplied arrays for each element in the docuemnt
  -- in the keys array it will store the path of the element where
  -- / is used to separate childen. 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 flatXML.
  -- It will take the supplied key, interate 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 illustrating a typical scenario where the Dynamic Services PL/SQL interface 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 application connecting to the service engine. Refer to Chapter 5.1.2 on how to register an application into the Dynamic Services service registry.

For a more elaborate sample that makes use of the currency service, refer to the `demo/consumer/currency.sql` file.

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 bundled into a simple service package (see Figure 3–1) and modeled 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
  - One service developer organization descriptor XML file
  - One or more service developer contact descriptor XML files
  - One service interface specification request definition (.xsd) file
  - One service interface specification response definition (.xsd) file

  The service descriptor file also describes in its service body section how the four types of service adaptors are to be used to handle the submitted service request (input adaptor), adapt the XML service request to the communication protocol used by the remote service developer (protocol adaptor), determine execution flow (if desired) of a service (execution adaptor), and transform the raw response returned by the remote service developer into a service XML response (output adaptor).

A compound service package has everything a simple service package has plus 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 package and with the name MANIFEST (case-sensitive). 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 "/", it indicates the link is an absolute link with respect to the root of the current package, with the root interpreted to be the root of the directory structure represented by package.

6.1 Quick Start

You can quickly start developing your own service by following the steps described in this section that are necessary to build a simple HTTP service. Later, you can enhance your service after reading about some more advanced concepts in later 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 developer organization and contacts XML files (see Section 6.1.2).
3. Edit the service classification XML file (see Section 6.1.3).
4. Create your service request definition XML schema file (see Section 6.1.4).
5. Create your service response definition XML schema file (see Section 6.1.5).
6. Edit the Service descriptor file, including both the service header section and the service body section (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 contains the XML schema 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 into a new directory using your name, for example, `demo/services/myService`.

   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 as shown in Example 6–1.

On UNIX platforms:

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/sd
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
```

On NT platforms:

- **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/sd directory and rename SampleService to portfolio, SampleOrg.xml to YahooOrg.xml, and SampleContact.xml to YahooContact.xml
- **d.** Navigate to the portfolio directory and rename SampleService.xml to ypfl.xml and SampleServiceClassification.xml to ypflClass.xml

2. Update the MANIFEST file under the `demo/services/myServices` directory to contain the following line as shown in Example 6–2.

Example 6–2 Update the MANIFEST File

```
/www.yahoo.com/dServices/sd/portfolio/ypfl.xml
```

This step lets the service package point to the correct service descriptor file that you will edit soon. Notice that all paths, used in this document, are relative to the `demo/services/myService` directory.

6.1.2 Service Developer (Provider) -- Organization and Contacts XML Files

Recall that the `YahooOrg.xml` file and the `YahooContact.xml` file described previously, reside in the directory `/www.yahoo.com/dServices/sd/`. These files contain service developer information about the organization and contacts for this particular service.

1. 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 everything 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>
```

2. 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 everything 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

Recall that the `ypflClass.xml` file described previously resides in the directory `/www.yahoo.com/dServices/sd/portfolio/`. 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 everything 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 and that 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.

1. Start by looking at a typical HTML Form. Example 6–6 shows a snippet 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">&nbsp;
  ...
</form>
```

The form takes one input called "SymbolList" and a HTTP GET Request is made to http://quote.yahoo.com/q when you click the submit button. An HTML page returns the quotes of the symbols that are specified in the input called "SymbolList".

2. **How do you make this into a service in the Dynamic Services Framework?**

   Next, notice that it takes one input called "SymbolList". From there you can expect the consumer (application) to pass in only one argument and generate an XML Schema file for our request, such as shown in Example 6–7.

Example 6–7 Generate an XML Schema File for the 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 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 that we have 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.

3. Create your request XML file and place it in the directory `/www.yahoo.com/dServices/sd/portfolio/` and name it `ypfl_req.xsd`.

6.1.5 Service Interface Specification - Response Definition

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

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

Example 6–8 Examine the Code Example 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>
```

2. **How do you transform that HTML into an XML that a service consumer (application) can use?**

   Determine what useful information should be extracted.

3. 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 that the input should be specified in -->
  <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 following the previous schema.

4. Create your response XML file and place it in the `/www.yahoo.com/dServices/sd/portfolio/` directory and name it `ypfl_resp.xsd`.

6.1.6 Editing the Service Descriptor

Next, the steps to modify the service descriptor ypfl.xml in the directory `/www.yahoo.com/dServices/sd/portfolio/` are described. The beginning of the service descriptor, with namespaces "sd" for all service descriptor tags and "xlink" for all of your document links to 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 that 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 above 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 above 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 above 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 won’t put Yahoo here
                because maybe other organizations can have the same kind of service
                which can be used in a fail-over scenario. -->
           <sd:NAME>PortfolioService</sd:NAME>
           <!-- This is mandatory; point this to the XML Schema file that you
                created above 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 above 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 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 maps the XML Requests that the 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 specifications contain information on how to map the aliases defined in Example 6–13 to the actual HTTP GET request.

Modify your service descriptor to contain the following 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 section, 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, refer to 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, refer to Section 6.3.2.

6.1.6.2.4 Output Adaptor Specifications The output 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 described previously.

Modify your service descriptor to contain the following as shown in Example 6–15.

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 Schema -->
                      <pflResp:Quote>
                        <!-- Also fully scope this -->
                        <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 HTML that returns into a more XML-compliant XHTML format, then applies the supplied XSL stylesheet to that XHTML document to form an XML document that conforms to the Response XML Schema that you previously defined in Section 6.1.5.

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

1. Launch 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 `demo/services/myService`.
2. Build a sample request following the syntax described in Section 6.1.4.
3. Use the DSAdmin utility again 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 select Change (C) to change the debug output levels, and finally select TRACE (3) to turn it to trace, so that you can see every step of the execution flow.

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

Section 6.2 through Section 6.5 provide more detailed description for each of the sections described in the service descriptor. Refer to these sections to create more advanced services.

6.2 Creating Advanced Services - Service Package

The service package is modeled 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 property files needed by the service.

The manifest file is expected to be found in the root directory of the package and with the name MANIFEST (case-sensitive). 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 "`/`", it indicates the link is an absolute link with respect to the root of the current package, with the root interpreted to be the root of the directory structure represented by package.

Note: The `<ORACLE_HOME>/ds/etc/xsd` directory contains the XML schema for the service descriptor and supplied adaptors. Refer to the files in this directory for more information.

6.3 Creating Advanced Services - Service Descriptor

A service 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, at least in part, if not specified in other documents to which the descriptor refers. There are two sections of the descriptor:

- Service header that describes the higher level behavioral descriptions of the service
- Service body that describes the implementation details of the service

Each of these sections, described in Section 6.3.1 and Section 6.3.2 maps to homonymous XML elements in the service descriptor. For service descriptor examples, refer to the supplied sample services under the *`<ORACLE_HOME>`*`/ds/demo/services` directory.

6.3.1 Service Header

The service header contains high-level descriptions of the service. For the most part, information specified here is descriptive and non-interpretive for browsing and documentation purposes. The exceptions are the service interface specification and the service identifier.

6.3.1.1 Naming Specification

Naming information contains a globally unique identifier as well as short and long descriptions of what the service does. Each service is addressed through an absolute name specified using the Universal Resource Naming (URN) conventions. Example 6–17 shows a sample naming specification section.

Example 6–17 Sample Naming Specification

```
<!-- Naming section provides identification information about the
     service. Among the elements, ID is the important one, which
     serves as unique identification. It must be an 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 svc.</sd:DESCRIPTION>
</sd:NAMING>
```

6.3.1.2 Package Specification

The service header 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 developer information section (see Section 6.3.1.3), this bit of information is critical for service maintenance. Example 6–18 shows a sample package specification section.

Example 6–18 Sample Package Specification

```
<!-- Package provides versioning 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 only used in the special case that you
         have custom java classes. Skip the whole JAR_POINTER
         section if there is no custom java classes 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 Provider (Service Developer) Information -- Organization and Contacts

High-level information about the service developer can be specified, including the provider’s company name, copyright information, and company URL. Detailed information drills down further into 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 Provider Information Specification.

Example 6–19 Sample Provider Information Specification

```
<sd:PROVIDER>
  <!-- Organization is a mandatory document that gives information
       about the 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 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

A set of deployment properties is comprised of suggestions from the service developer to aid the service engine administrator during registration time. They 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 in this section 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 descriptor. Example 6–20 shows a sample deployment specification.

Example 6–20 Sample Deployment Specification

```
<sd:DEPLOYMENT>
  <!-- Follow our convention in pathname within the zip file -->
  <sd:CLASSIFICATION
    xlink:href="/www.oanda.com/dServices/currency/class.xmll"/>
  <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 7.1 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-Schemas is specified by URLs: when a relative URL is used, that is, relative to the service package submitted by the service developers. By specifying these schemas, the service developer enforces the syntax in which consumers (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 consumer (application) sends a request, before the actual service developer is contacted.

The service developer 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 necessary exception) definition. The engine also exposes to consumers (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 the Dynamic Services framework, class generators can be used to create Java classes that map to the request/response XML-Schemas.

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

The service body contains more detailed descriptions and information that is used by the Dynamic Services engine at service execution time. Specifically, it is sectioned into details as 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 remote service developer
- Execution determines the execution flow of a service
- Output transforms the raw response returned by the remote service developer into a service XML response.

Figure 6–1 shows a sample service execution and the role 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

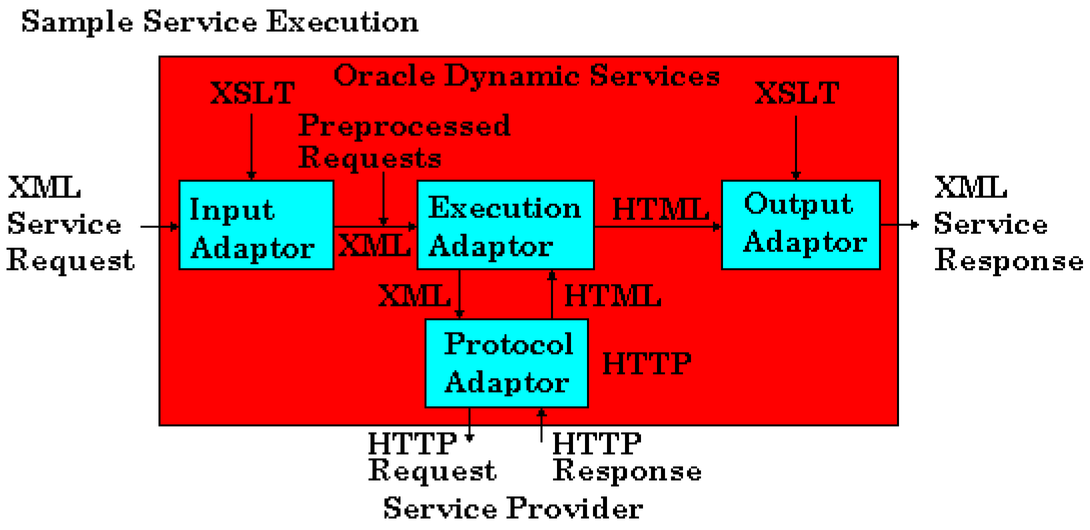
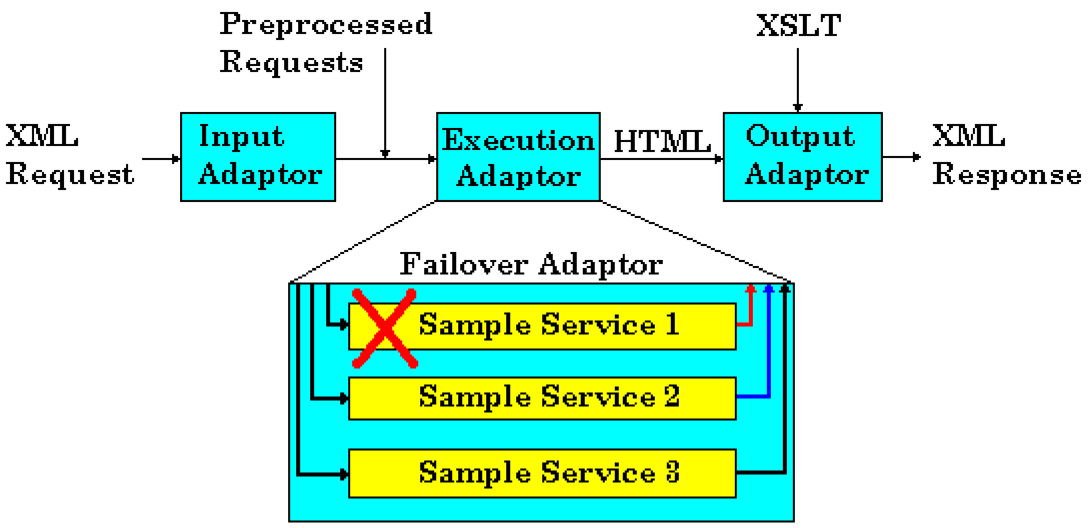


Figure 6–2 shows a sample execution adaptor and the role the execution adaptor plays in identifying the way in which one or more sample services is to be executed. In this case, an execution adaptor would specify in its execution flow logic how and why a set of one or more sample services is to be executed. 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 fail to execute. In this example, 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 developer can specify an adaptor in the specification that needs to be used at each of these layers. A set of pre-built customizable adaptors comes prepackaged with Dynamic Services.

6.3.2.1 Input Handling and Adaptor Specifications

The input section specifies the list of necessary as well as optional processing on the request that is submitted by the consumer (application). This includes the following sections:

- 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 sections.

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

Example 6–22 Sample Namespace 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 developers may specify additional directives for the purposes 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 communication protocol of the service. For example, for HTTP, specifying a XPath into the request XML 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 value:

- Supplying an Xpath, which at service execution time, is applied to the XML service request 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.
- Supplying a consumer application profile property and optionally its modifier. The corresponding value is fetched dynamically at execution time according to the identity of the consumer (application).

Example 6–23 shows a sample aliases section, the first using the XPath approach for a value, using the namespace described above, 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>SomeFragment</sd:NAME>
    <!-- indicates that the value is obtained dynamically from the user
         request following the namespace specifed by "fb" above -->
    <sd:VALUE>{@xpath:fragment=/fb:foobarReq/fb:frag1}</sd:VALUE>
  </sd:ALIAS>
  <sd:ALIAS>
    <sd:NAME>AccountNumber</sd:NAME>
    <!-- indicates that the value is obtained dynamically from the user
         request following the namespace specifed by "fb" above -->
    <sd:VALUE>{@xpath:value=/fb:foobarReq/fb:param1}</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 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 section can optionally specify an adaptor that further processes the service request before sending it to the service developer. 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 developer has the option of specifying some adaptor specific parameters in the PARAMETERS element under the adaptor section 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 consumer (application) submits or sends to the service engine is validated with the Input XML-Schema that is specified previously in the header. However, a service developer can optionally supply some form of Schema Mapping specifications (for example, through a XSL Transformation) that could map this

Input XML-Schema to a presentation form such as HTML or Wireless Markup Language (WML). As a result, the consumer (application) can easily provide to its clients a way to input service requests, for applications that have an HTML or WML interface. Notice that the engine is not responsible for the rendering: all that the engine is responsible for is the capabilities to store and retrieve the mapping. The engine only provides the mapping(s) of the transformation. The actual transformation is done on the consumer (application) side by the client toolkit. If you have a mapping of the schema into an HTML form, the consumer (application) can use the mapping to render the Input schema to an HTML form for their Web application. The consumer (application) can then transform the HTTP Requests back to an XML document that conforms to the XML-Schema specified by the service developer. Finally, the request XML is sent to the service engine formulating a service request.

6.3.2.2 Protocol Adaptor Specifications

The protocol section identifies the way that the service 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 that handles the communication to the underlying service. This class name is either found in the service package given by the service developer during registration or in the set of libraries that the service engine provides. A driver specification can also be specified to make sure that a certain class is in the classpath for the Adaptor to function properly. Finally, for the specified adaptor, the service developer has the option of specifying some adaptor specific parameters in the PARAMETERS element under the adaptor section, which are validated at service registration time and interpreted at runtime by the adaptor. For example, for HTTP, it 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 Specifications

The execution section identifies the way in which the service has to be executed. Its responsibility is to take in the request XML and return the response from the underlying service developer. The default execution adaptor is a standard simple adaptor that follows the path described previously. There can also be complex 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 that performs the execution. For the specified adaptor, the service developer has the option of specifying some adaptor specific parameters in the PARAMETERS element under the adaptor section, which are validated at service registration time and interpreted at runtime by the adaptor.

The result of the execution adaptor is the response given back from the service. If the service is a simple service, the response will be in the native format of the service developer. 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 developer will use the default prepackaged simple adaptor. For complex and compound service execution, the service developer can leverage the supplied compound execution adaptor.

6.3.2.4 Output Handling and Adaptor Specifications

The output section specifies the list of necessary as well as optional processing to produce the service response to the consumer (application). This includes the following sections:

- Output adaptor
- Rendering directives

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

6.3.2.4.1 Output: Adaptor Specification The output section can specify 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. This output adaptor specification is a fully qualified name of the class implementing the `oracle.ds.engine.OutputAdaptor` interface that handles the transformation. For the specified adaptor, the service developer has the option of specifying some adaptor specific parameters in the PARAMETERS element under the adaptor section, 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 developers will either use the prepackaged adaptors, such as the XSLT adaptor. For compound services, service developers can use no adaptor because the response from the execution adaptor is often in the proper format prescribed by the Output XML-Schema.

6.3.2.4.2 Output: Rendering Directives As far as the service execution flow is concerned, the output section is the final stop. However, Dynamic Services also provides additional mechanisms for the service developer to optionally specify mappings (for example, in the form of XSL transforms) that map this response XML to other forms, such as HTML or WML. Consumers (applications), rather than service engines, are responsible for making use of the transformation to render the desirable output. Dynamic Services merely provides a means to store it and make it accessible from the 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 prepackaged in the Oracle Dynamic Services distribution.

6.4.1 Input Adaptor

Section 6.4.1.1 describes the input adaptor prepackaged in the Oracle Dynamic Services distribution.

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` and the INPUT/ADPTOR/PARAMETERS element in the Service Descriptor must contain the following as 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 XSL Stylesheet specified can make use of the aliases defined as xsl variables in the following ways as shown in Example 6–26.

Example 6–26   Sample Aliases Defined as xsl Variables

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

In the DSXSLTInputAdaptor there is also an option of bringing in other service descriptors before applying the stylesheet. This can be done with 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 InputAdaptor specification for DSXSLTInputAdaptor, refer to the YahooPortfolio service.

6.4.2 Protocol Adaptors

Section 6.4.2.1 through Section 6.4.2.4 describes the protocol adaptors prepackaged in the Oracle Dynamic Services distribution.

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

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

The parameters and other details 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 name space 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 section 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 some mandatory elements as well as some optional ones. 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> 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 everytime 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 the following two possible structures. The first one put 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 Based64. The login name and password can be aliases that refer to other sources as shown in Example 6–28 .

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. The lifetime of cookies is within a service consumer (application) session.

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

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

The implementation is based on Sun Microsystems JSSE 1.0.1 reference implementation, which is intended to be used for non-commercial use.

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

DSJDBCProtocolAdaptor processes the JDBC/2.0, sets up the JDBC/2.0 connection based on the request, and returns a handle to process the raw output from the database.

Example 6–29 shows the parameters and other details to execute DSJDBCProtocolAdaptor are 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 name space 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:connection>
          <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>
        <xsql:dml>
          insert into {@TableName} values (, …)
        </xsql:dml>

        <xsql:query>
          select {@ColName1},{@ColName2} from {@TableName} where }
        </xsql:query>
          </jpParams:JDBC_PA_PARAMS>
        </sd:PARAMETERS>
      </sd:ADAPTOR>
   </sd:PROTOCOL>
```

This section consists of two or more parts, one is to set up the JDBC connection, the others are queries or DML actions or both. Since the implementation of DSJDBCProtocolAdaptor employs the Oracle xsql package, xsql's name space must be included in the JDBC_PA_PARAMS elements. The usage of <username>, <password> <dburl>, <driver> and <autocommit> are consistent with those defined in Oracle JDBC Driver 2.0.

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

DBService is a sample service to illustrate how to execute any actions to a database.

When a service using DSJDBCProtocolAdaptor is executed within a service consumer (application) session, the JDBC connection (identified by username, password and dburl) is reused within the session. At session closing time, the JDBC connection is rolledback. 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 a HTTP connection.

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

DSSMTPProtocolAdaptor is the protocol adaptor used for service developers that use SMTP as an underlying service access mechanism; such services can include simple mail sending services that get invoked upon an error.

Example 6–30 shows the parameters and other details to execute DSSMTPProtocolAdaptor are defined in the service descriptor, in the section SERVICE_BODY / PROTOCOL / ADAPTOR. 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, 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>
          <!-- Pre-defined XML Schema of the SMTP Protocol Adaptor Params -->
          <spParams:SMTP_PA_PARAMS
            xmlns:spParams="http://www.oracle.com/ds/2000/SMTP_PA_PARAMS">
            <spParams:Host>gmsmtp01.oraclecorp.com</spParams:Host>
            <spParams:Port>25</spParams:Port>
            <spParams:From>egroup@lackey.us.oracle.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 this example, 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 describes the execution adaptors prepackaged in the Oracle Dynamic Services distribution.

6.4.3.1 oracle.ds.engine.ea.DSFailOverExecutionAdaptor

DSFailOverExecutionAdaptor takes as PARAMETERS an ordered list of compatible services, which means they respond to the same service interface. At execution time, it tries to execute the first one in the list, if it fails it moves to the second one, and so on until it finds a service that executes with no exception. If none succeed, 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 Service 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 snippet of a compound service execution adaptor specification.

Example 6–32 Sample Compound Service Specification

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

Compound services allow you to 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 developer 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 is a list of 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 outputs another array of messages comprised of responses returned by the service execution(s). There are two possible syntaxes 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 executing the service 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 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 performs transformations of service messages, as either requests or responses. It takes an XSLT as a property in its Properties element and applies that XSLT on all the incoming service messages to produce another list of outgoing services messages. Example 6–35 shows a sample module specification of the MessageTransformer.

Example 6–35 Sample MessageTransformer Module

```
<ceParams:Module name="ID4">
  <ceParams:Class>
    oracle.ds.engine.ea.compound.MessageTransformer
      </ceParams:Class>
      <ceParams:Properties>
        <ceParams:XSLT>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 is specified in an element called XSLT and the stylesheet is able to transform the starting service message into a well known structure described in Example 6–36 so that a list of service messages can 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 using the SingleTransformation option.

Example 6–37 Sample MessageSplitter Module Using the Single Transformation 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>
```

- MultipleTransformations

  With this option, a list of XSLTs is specified, corresponding to a list of transformation to the original service message to arrive at a list of resulting messages. With each XSLT, an index is also specified to order the list of service message that result in the one-by-one application of the XSLT stylesheets. Example 6–38 shows a sample module specification of the MessageSplitter 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>
```

oracle.ds.engine.ea.compound.MessageMerger

This module merges multiple service messages into one single message in the following form shown in Example 6–39 and then applies an XSLT transformation on that message.

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 wrapped into a new XML element 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 expects 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

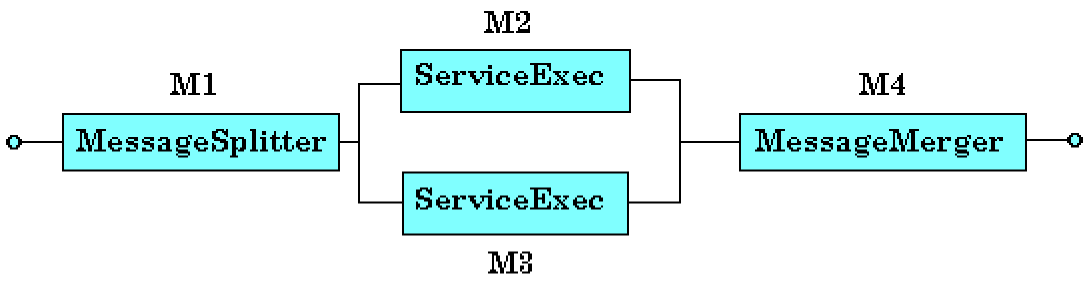


Figure 6–3 shows a possible usage of the modules described previously. In this case, you can achieve the parallel execution of a couple of services, and to do that you

first split the consumer (application) supplied request, and finally join the two services responses into one single response. In this example, 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 inputs of each of the modules. DSCompoundServiceExecutionAdaptor makes use of this information to coordinate the execution flow and creates the necessary threading 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.

6.4.3.3 oracle.ds.engine.ea.DSConditionalExecutionAdaptor

DSConditionalExecutionAdaptor controls the flow of execution in that it executes a specified service based on the value of a certain alias that has been defined. Service developers 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 referencd, thus the value of this attribute should just be the alias name rather than {@alias}, which denotes preprocessing it 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 either be an "execute" element, which denotes that you have reached a leaf and that its element value is the ID of the service to be executed, or 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 only referencing it so the syntax used is just the alias
             name and not {@alias} -->
        <deParams:switch on="eventType">
          <deParams:case value="CONNECT">
            <!-- Here’s 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 adaptors prepackaged in the Oracle Dynamic Services distribution.

6.4.4.1 oracle.ds.engine.ioa.DSXSLTOutputAdaptor

DSXSLTOutputAdaptor applies an XSLT transformation to either a java.net.URLConnection or an oracle.xml.parser.v2.XMLDocument raw response. In the case of a URLConnection, it first checks if the content-type is 'text/html', 'text/xml', or 'application/xml'. In the HTML case, it first applies a transformation to convert the HTML representation into XHTML, compliant with W3C XHTML 1.0 specifications. Finally, an XSLT 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. Refer to the Yahoo portfolio sample service for an example.

6.5 Creating Advanced Services - Building Your Own Adaptors

Service developers can supply their own adaptors. As described previously, according to the layer that the adaptor addresses during the service execution, each 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, refer to the description in the JavaDoc.

6.5.1 Packaging

Once you have built your classes and grouped them into a jar and bundled it into your service package, ensure the jar pointer element in the binary resource section of the service descriptor refers to the correct jar. 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 identity, and how to test a service execution. Other simple tasks such as unregistering a service, adding user properties, and so on can also be performed using the DSAdmin utility. In this chapter, a brief overview of other topics relevant to administrators is provided.

7.1 Service Response Caching

The Dynamic Services engine uses the Oracle8*i* 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.

7.2 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 procedure to periodically clean up the cache. A procedure is supplied within the DS_CacheManager package to start the DBMS_JOB procedure that performs the cache clean up. This procedure is called startCleanupJob, and it takes a VARCHAR2 argument that specifies the interval between cleanup jobs.

7.3 Connecting Multiple Dynamic Services Engine Instances

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 DynamicSservices engine includes:

- A registry mirror in an Oracle8*i* 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 `resync` of the registry.

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 `resync`.

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

7.4 Additional Operations of the DSAdmin Utility

You can browse through the DSAdmin utility to find additional administrative tasks that may be useful to perform. There is an entire menu on service consumer management.

7.4.1 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

```
<ORACLE_HOME>/ds/bin/dsadmin -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 token strings, 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.

7.5 Management Console

This release contains a technology preview of the management console. A simple Web-based interface is provided for browsing and running services.

See Section 4.7 for information on installing the management console.

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 developers defining a new service may be able to define new adaptors.
- During service registration, the service descriptors are not XMLSchema-validated.

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`

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:

$ORACLE_HOME/ds/doc/dsfaq.txt

C

Descriptive Matrix

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

C.1 Syntax of the Service Descriptor Schema

At the top level, a SERVICE_DESCRIPTOR schema contains the following data as shown in Table C–1. Required data is designated by bold field names.

Table C–1 Descriptive Matrix of the Service Descriptor Schema

Field Name	Description	Data Type	Length
SERVICE_DESCRIPTOR	This is the root element in the service descriptor document. A service provider will define a service by writing an XML document that complies to the XML schema file sd.xsd and other auxiliary documents such as the classification, organization, and contact XML documents.		
SERVICE_HEADER	The service header section. Contains high level descriptions of the service.		
NAMING	The naming section. Contains a globally unique identifier ID, as well as NAME and DESCRIPTION fields describing what the service does.		
ID	Uniquely identifies the service and must be a Uniform Resource Name (URN).	Uri	
NAME	The name of the service.	String	
DESCRIPTION	The description of the service.	String	

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

Field Name	Description	Data Type	Length
PACKAGE	The package section. Contains version specifications and pointers to where the service update is to be performed.		
VERSION	The version number of the service definition package.	String	
RELEASEDATE	The release date of the service definition package.	Date	
UPDATEURL	The URL to obtain the latest version of the service definition package. In general, a service should contain an update URL. However, for services created by an administrator, it is meant to be used locally, UPDATEURL is not applicable. (Not used currently.)		
BINARY_RESOURCES	The binary resources section. For advanced usage, such as specifying a java class file or specifying a stylesheet for custom services and adaptors.		
JAR_POINTER	The URL of the jar containing service specific Java classes and resources within the service definition zip file.		
EXCEPTIONS	The exceptions section. Contains the specification for the resource bundle for custom exceptions.		
EXCEPTION_MSG_BUNDLE	The specification for the custom exceptions that rely on the custom resource bundle.	String	
DEPLOYMENT	The deployment section. Contains a set of deployment properties from the service developer to aid the service administrator during service registration.		
CLASSIFICATION	This is 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 be compliant to sd_classification.xsd. See Table C–2 for a description of the classification schema.		
CACHING	The caching section. Contains recommended caching parameter values.		
MAX_AGE	MAX_AGE is 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.	Nonnegative Integer	
SESSION_PRIVATE	SESSION_PRIVATE is true means that the scope of the cache entry is within the service engine user session. The default value is false.	Text Boolean	

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

Field Name	Description	Data Type	Length
USE_PROTOCOL	USE_PROTOCOL is true means that the protocol caching parameters will override MAX_AGE. The default value is false.	Text Boolean	
PROVIDER	The service provider section. Contains information about the service developer including the provider's company name, copyright information, company URL, contacts for support, and URLs for logos		
ORGANIZATION	This is an xlink to the organization XML document. Provides generic information about the service provider. The xml file should be complaint to sd_organization.xsd. See Table C–4 for a description of the organization schema.		
CONTACTS	The contacts section. Contains detailed support contacts for this service.		
CONTACT	This is an xlink to the contact XML document. Provides information to contact a person for any issues related to the service. The xml file should be compliant to sd_contact.xsd. See Table C–3 for a description of the contact schema.		
INTERFACE	The service interface specification. Contains the definition of an interface characterized by the schema specifications of its input, output, and exceptions.		
NAME	The name of the interface template.	String	
INPUT_SCHEMA	This is an xlink to the request XML document.The URL to the request definition XML schema document or DTD defining the XML service request syntax.		
OUTPUT_SCHEMA	This is an xlink to the response XML document.The URL to the response definition XML schema document or DTD defining the XML service response syntax.		
SERVICE_BODY	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.		

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

Field Name	Description	Data Type	Length
INPUT	The input section. Contains the details in preprocessing the XML request from the service consumer and includes the following sections: namespaces, alias directives, input adaptor, and rendering directives.		
NAMESPACES	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	The namespace section. The namespaces definitions will be used for the xpath definitions in the values of aliases.		
PREFIX	The namespace prefix.	String	
VALUE	The namespace value.	String	
ALIASES	The aliases section. Used by service developers 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	The alias section.		
NAME	The alias name.	String	
VALUE	The alias value. Can be specified as an Xpath or as a consumer application profile property and optionally its modifier. The Xpath is used at runtime 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.	String	
RENDERERS	The renderers section. Contains additional rendering directives. The service developer can optionally supply some form of schema mapping specifications, such as an XSL transformation, that could map this Input XML-schema to a presentation form such as HTML or Wireless Markup Language (WML). Thus, the consumer application can provide to its clients a way to input service requests, for applications that have an HTML or WML interface.		

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

Field Name	Description	Data Type	Length
ADAPTOR	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	The fully qualified name of the class implementing the oracle.ds.engine.InputAdaptor Java interface that handles the processing.	String	
PARAMETERS	The parameters specification. The service developer 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.		
PROTOCOL	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.		
ADAPTOR	The adaptor section.		
NAME	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.	String	
DRIVER	The driver specification. Ensures that a certain class in the classpath for the Adaptor to function properly.	String	
PARAMETERS	The parameters specification. The service developer 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.		
EXECUTION	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 developer.		
ADAPTOR	The adaptor section.		

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

Field Name	Description	Data Type	Length
NAME	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.	String	
PARAMETERS	The parameters specification. The service developer 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.		
OUTPUT	The output section. Specifies the list of necessary as well as optional processing to produce the service response to the consumer (application). Includes the following sections: output adaptor and rendering directives.		
RENDERERS	The renderers section. Contains additional rendering directives. The service developer can optionally supply some form of schema mapping specifications, such as in the form of XSL transforms, that map this response XML to other forms, such as HTML or WML. Thus, the consumer application can provide to its clients a way to output service responses, for applications that have an HTML or WML interface.		
ADAPTOR	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	The fully qualified name of the class implementing the oracle.ds.engine.OutputAdaptor Java interface that handles the transformation.	String	
PARAMETERS	The parameters specification. The service developer 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.		

At the next level, a CLASSIFICATION schema contains the following data as shown in Table C–2. Required data is designated by bold field names.

Table C–2 Descriptive Matrix of the Classification Schema

Field Name	Description	Data Type	Length
CLASSIFICATION	The classification information from the service provider to assist the service administrator during registration.		
CATEGORY	The hierarchical categories specified using the substring of the Distinguished Name (DN) specified in the RFC2253 specification (http://www.ietf.org/).		
KEYWORDS	The keywords separated by commas.		

At the next level, a CONTACT schema contains the following data as shown in Table C–3. Required data is designated by bold field names.

Table C–3 Descriptive Matrix of the Contact Schema

Field Name	Description	Data Type	Length
CONTACT	The contact information from the service provider or service developer.		
NAME	The name of the contact.	String	
EMAIL	The email address.	String	
PHONE	The phone number.	String	
FAX	The FAX number. The default is " ".	String	
PAGER	The Pager number. The default is " ".	String	
MOBILE	The mobile number. The default is " ".	String	

At the next level, an ORGANIZATION schema contains the following data as shown in Table C–4. Required data is designated by bold field names.

Table C–4 Descriptive Matrix of the Organization Schema

Field Name	Description	Data Type	Length
ORGANIZATION	The company information from the service provider.		
NAME	The provider's company name.	String	
COPYRIGHT	The copyright information for the company.	String	
URL	The URL for the company.	URI	
LOGOURL	The URL for company's logo.	URI	

C.2 Syntax of the Parameters Section for the Packaged Adaptors

C.2.1 oracle.ds.engine.ioa.DSXSLTInputAdaptor

An Input Adaptor schema contains the following data as shown in Table C–5. Required data is designated by bold field names.

Table C–5 Descriptive Matrix of the Input Adaptor Parameters

Field Name	Description	Data Type	Length
XSLT_IA_PARAMS	The parameters section of the input adaptor prepackaged in the Oracle Dynamic Services distribution.		
XSLT	The XSL Stylesheet used by the input adaptor.		

C.2.2 oracle.ds.engine.ioa.DSXSLTOutputAdaptor

An Output Adaptor schema contains the following data as shown in Table C–6. Required data is designated by bold field names.

Table C–6 Descriptive Matrix of the Output Adaptor Parameters

Field Name	Description	Data Type	Length
XSLT_OA_PARAMS	The parameters section of the output adaptor prepackaged in the Oracle Dynamic Services distribution.		
XSLT	The XSL Stylesheet used by the output adaptor.		

C.2.3 oracle.ds.engine.pa.DSHTTPProtocolAdaptor

An HTTP Protocol Adaptor schema contains the following data as shown in Table C–7. Required data is designated by bold field names.

Table C–7 Descriptive Matrix of the HTTP Protocol Adaptor Parameters

Field Name	Description	Data Type	Length
HTTP_PA_PARAMS	The section for the HTTP Protocol Adaptor parameters.		
Method	The method used for the HTTP request. Must be one of three options: GET, POST, or HEAD.	String	
URL	The URL to be contacted.	String	
RequestHeaders	Setting HTTP Request Headers in the request. Contains the definition of additional HTTP request headers that the user wants to define.		
RequestHeader	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.	String	
QueryStringParameters	Contains the definition of the Query string used for complex GET or POST requests.		
QueryStringParameter	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.		
Authorization	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.		
EncodedString	The encoded string that contains the user’s login name and password.	String	
Credential	The credential section. Contains the username and password elements.		
Username	The user’s login name.	String	
Password	The user’s login password.	String	

C.2.4 oracle.ds.engine.pa.DSJDBCProtocolAdaptor

A JDBC Protocol Adaptor schema contains the following data as shown in Table C–8. Required data is designated by bold field names.

Table C–8 Descriptive Matrix of the JDBC Protocol Adaptor Parameters

Field Name	Description	Data Type	Length
JDBC_PA_PARAMS	The section for the JDBC Protocol Adaptor parameters.		
Connection	The connection section. Specifies the database connection parameters.		
Username	The username.	String	
Password	The password.	String	
dburl	The database URL.	String	
driver	The name of the Oracle JDBC driver	String	
autocommit	The autocommit parameter value. Specified as true or false. The default is true to automatically commit any updates.	Boolean	

C.2.5 oracle.ds.engine.pa.DSSMTPProtocolAdaptor

A SMTP Protocol Adaptor schema contains the following data as shown in Table C–9. Required data is designated by bold field names.

Table C–9 Descriptive Matrix of the SMTP Protocol Adaptor Parameters

Field Name	Description	Data Type	Length
SMTP_PA_PARAMS	The section for the SMTP Protocol Adaptor parameters.		
Host	The SMTP host name.	String	
Port	The SMTP port number.	Positive Integer	
From	From whom the mail was sent.	String	
To	To whom the mail is to be sent.	String	
cc	To whom else should receive a copy of the mail.	String	
bcc	A copy of the mail is to be sent to the sender of the mail.	String	

Table C–9 Descriptive Matrix of the SMTP Protocol Adaptor Parameters (Cont.)

Subject	The subject line of the mail.	String	
MsgHeaders	The message header section.		
MsgHeader	The header of the mail message. It is specified as a "name" attribute and element value of this element.		
MsgBody	The body of the mail message.		

C.2.6 oracle.ds.engine.ea.compound.DSCompoundServiceExecutionAdaptor

A Compound Execution Adaptor schema contains the following data as shown in Table C–10. Required data is designated by bold field names.

Table C–10 Descriptive Matrix of the Compound Execution Adaptor Parameters

Field Name	Description	Data Type	Length
COMPOUND_EA_PARAMS	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	The modules section. Contains a set of modules.		
Module	The module section.		
Class	The class type. One of four classes of compound service execution modules: service execution, message transformer, message splitter, and message merger.	String	
Properties	The property type. For a compound service class type, there are two possible syntaxes for its properties: executeSingleRequest or executeAllRequests.		
Message Splitter Properties: choice of MultipleTransformations or SingleTransformation			
MultipleTransformations	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	The specified list of XSLTs corresponding to a list of transformations to the original service message to arrive at a list of resulting messages.		

Table C–10 Descriptive Matrix of the Compound Execution Adaptor Parameters (Cont.)

SingleTransformation	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	The specified XSLT.		
Service Execution Properties: choice of executeSingleRequest or executeAllRequests			
executeSingleRequest	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 and executing the service specified by the ID with that request.		
executeAllRequests	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 building up a response event.	String	
Message Merger Properties			
XSLT	The message merger XSLT.	String	
Message Transformer Properties			
XSLT	The message transformer XSLT.	String	
Execution Flow Definition Using a Dependency Matrix			
Graph	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	The row's name attribute specifies the associated module.		
column	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.	Non Negative Integer	

C.2.7 oracle.ds.engine.ea.DSConditionalExecutionAdaptor

A Conditional Execution Adaptor schema contains the following data as shown in Table C–11. Required data is designated by bold field names.

Table C–11 Descriptive Matrix of the Conditional Execution Adaptor Parameters

Field Name	Description	Data Type	Length
CONDITIONAL_EA_PARAMS	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 where the leaf elements are the IDs of the service to execute.		
switch	The switch element. The "on" attribute specifies the alias name on which the switch is based.		
case	The case element. The value attribute specifies the value of the switch.		
execute	The execute element.	String	
default	The default element.		
execute	The execute element.	String	

C.2.8 oracle.ds.engine.ea.DSFailOverExecutionAdaptor

A Failover Execution Adaptor schema contains the following data as shown in Table C–12. Required data is designated by bold field names.

Table C–12 Descriptive Matrix of the Failover Execution Adaptor Parameters

Field Name	Description	Data Type	Length
FAILOVER_EA_PARAMS	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	The execute element. The priority attribute defines the priority among the services.		

Glossary

compound service package

Contains multiple services and includes one additional file, a jar file, which contains all Java classes and property files needed by the compound service package.

Distinguished Name (DN)

The unique name of a directory entry in Oracle Internet Directory. It comprises all of 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 Service

A component within the Internet computing model that delivers a specialized value-added functionality. 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 Services.

Dynamic Services framework

An open Java-based programmatic framework for enhancing Oracle8*i* as the Internet platform for incorporating, managing, and deploying Internet services. It comprises dynamic services engine, a set of Dynamic Services, as well as users of Dynamic Services (service consumers or applications). 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.

execution adaptor

A routine that executes a service request in a particular flow. A flow could be as simple as relaying the request to contacting a service developer, or as complicated as relaying a request to a service developer and relaying the response to another service developer.

input adaptor

A routine that post-processes the service input from consumers (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 to the standard service response.

protocol adaptor

A routine that transforms the standard service request to the inputs needed by the underlying service following the underlying protocol.

registry

service

A component within the Internet computing model that delivers a specialized value-added functionality.

service administrator

The person who performs administrative activities for the Dynamic Services engine, such as enabling or disabling of services, tuning caching parameters of a service, and so forth.

service consumer

An application that utilizes Dynamic Services to aggregate services from service developers and provides a dynamic service to their customers.

service descriptor

A descriptor defining the behavior of a service, containing service developer information, description of service features, service management information, service input adapter, service output adapter, and other provider-specific sections, such as secure access, caching parameters, and so forth.

service developer

A business partner or application developer that provides and manages the content of a service for the Dynamic Services execution engine; typically, the service developer is the owner of some data resource or process, such as, the owner of a currency exchange rate Web site.

simple service package

A service is bundled into a simple service package (see Figure 3–1) and modeled 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 and .xsd files:

- One classification descriptor XML file
- One service developer organization descriptor XML file
- One or more service developer contact descriptor XML files
- One service interface specification request definition (.xsd) file
- One service interface specification response definition (.xsd) file

service provider

Someone who provides content for a service.

Index