BEA AquaLogic® Data Service Platform

Administration Guide

Version: 3.0
Revised: April 2008

Contents

1. Overview of ALDSP Administration

2. Getting Started with ALDSP Administration

3. Deploying Dataspaces

4. Configuring ALDSP Resources

5. Securing ALDSP Resources

6. Viewing Native Web Services

7. Viewing Metadata Using the Service Explorer

8. Configuring Query Results Cache

9. Working With Audit and Log Information

10. Extending Database Support

CHAPTER 1

Overview of ALDSP Administration

This chapter introduces AquaLogic Data Services Platform (ALDSP) administration. It explains the concept of ALDSP-Enabled WebLogic domains and introduces the ALDSP Administration Console components.

The primary audience for this document is WebLogic Server and/or AquaLogic Data Services Platform administrators.

The chapter contains the following sections:

Note: ALDSP was previously named Liquid Data. Some artifacts of the original name remain in the product, installation path, and components.

Administering ALDSP

ALDSP is an integration software that unifies data programming by using data services. You can deploy it to WebLogic Server and administer tasks such as dataspace deployment, managing services accounts, controlling user access, and configuring runtime security through the ALDSP console.

Some administrative tasks can be performed through WebLogic console such as starting and stopping the server, configuring connection pools and data sources, logging, and so forth. The WebLogic Platform provides extensive tools and capabilities for configuring and maintaining a large-scale, production-level integration platform.

This section introduces you to the general administration tasks that you can perform using the ALDSP console. It includes the following topics:

- Securing Data
- Caching Query Results
- Viewing Metadata

For information on WebLogic administration, refer to System Administration for BEA WebLogic Server 9.2.

Securing Data

ALDSP leverages the security model of the WebLogic Platform to ensure data security. WebLogic uses security policies that control access to deployed resources based on user credentials or other factors.

Note: For information about securing a server see the WebLogic Server 10.0 document"Securing a Production Environment".

ALDSP enables you to apply policies to its data resources at various levels ranging from the dataspace to data elements. In addition, you can secure resources based on data values (called instance-level security). For example, you can secure objects if an element value exceeds a specific threshold.

For details, see Chapter 5, "Securing ALDSP Resources."

Caching Query Results

ALDSP can cache query results for data service functions to enhance overall system performance. Caching data alleviates the burden on back-end resource and improves data request response times from the client's perspective. If you want to cache data service function results, you must explicitly enable results caching in the ALDSP Administration Console.

For more information, see Chapter 4, "Configuring ALDSP Resources."

Viewing Metadata

Traditionally, enterprises have lacked a universal mechanism for advertising availability of data resources across source types, or for communicating information about those resources. ALDSP provides this capability through dynamically generated metadata.

Data service metadata serves these primary purposes:

- It helps developers create client applications that use the information made available by ALDSP by revealing what data is available and how to use it.
- It helps administrators maintain ALDSP by providing a mechanism to gauge effects of changes in underlying data sources upon a data service deployment.

Metadata provides information on data services such as their public functions, datatypes, data lineage, and more. It also provides *where used* information, showing dependencies between data services.

For more information, see Chapter 7, "Viewing Metadata Using the Service Explorer."

Understanding ALDSP-Enabled WLS Domains

An ALDSP domain is created and deployed on WLS 9.2 and is a collection of resources managed as a single unit. In case of ALDSP, the WebLogic Administration console is used to create users and assign roles for a domain. An ALDSP domain may constitute one or more dataspaces deployed on a WebLogic Server as well as clusters. It is also where you deploy the ALDSP dataspace for your domain.

The WebLogic Administration Console is a web-based interface for configuring and monitoring a WebLogic domain. In cases when the domain has more than one server, one of the servers is designated as the *Administration Server* for the domain. The Administration Server then serves as the central point of control for an entire domain.

If there is only one server in a domain, then that server is the Administration Server in addition to the other functions it provides. Any other servers in a domain are *Managed Servers*.

For more information about domains, see "Understanding WebLogic Server Domains" in *Configuring and Managing WebLogic Server*.

Understanding the Relationship between ALDSP and WebLogic Domains

ALDSP constitutes one or more dataspaces, which have a set of associated resources deployed on a WebLogic domain. To manage an ALDSP dataspace, start the WebLogic Server within the domain where an ALDSP dataspace is deployed, and then use the ALDSP Administration Console for that server to configure and manage ALDSP resources.

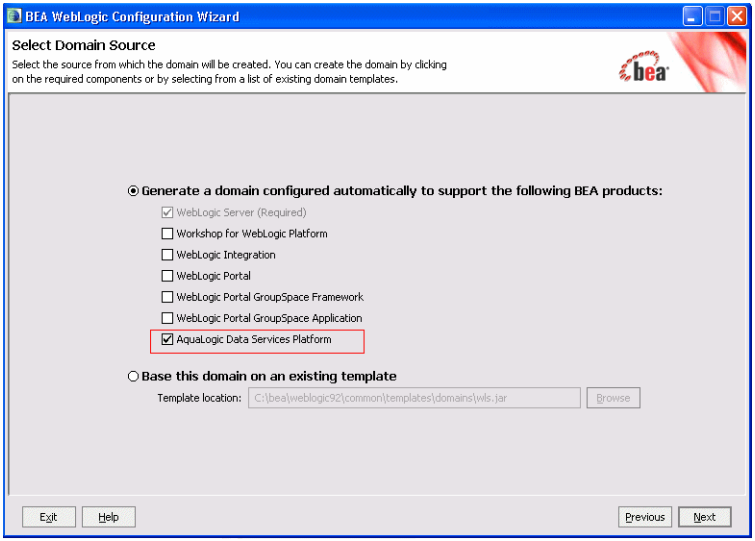
Creating a New Domain

A dataspace created in the ALDSP development environment, works with WebLogic domains that have been provisioned for ALDSP. You can use the BEA WebLogic Configuration Wizard to create such domains.

To create a new domain provisioned with ALDSP:

1. On Windows systems, choose Programs > BEA Products > Tools > Configuration Wizard.
2. In the wizard, choose AquaLogic Data Service Platform Domain as the domain source as shown in Figure 1-1.

Figure 1-1 Selecting ALDSP as the Domain Source



3. Follow the on-screen instructions to complete the initial configuration of the domain.

For more information on creating domains, see "Creating WebLogic Domains Using the Configuration Wizard" in the WebLogic Platform documentation.

Provisioning an Existing Domain for ALDSP

If you have an existing WebLogic Server domain and you want to setup ALDSP project within that domain, you can provision the domain for ALDSP, using the Configuration Wizard:

1. Open the Configuration Wizard:

   Start > Programs > BEA Products > Tools > Configuration Wizard

2. Select the option: Extend an existing WebLogic configuration.

3. Select the domain you wish to enable for ALDSP (such as: `AL_HOME/samples/domains/portal`).

4. Select AquaLogic Data Services Platform extension using the Extend my domain automatically to support the following added BEA Products option.

For information on selecting domain setting options see Creating WebLogic Domains Using the Configuration Wizard.

Once a domain is provisioned with ALDSP, you can deploy dataspace to WebLogic Server enabled for ALDSP.

For additional information see Chapter 3, "Deploying Dataspaces."

Understanding Console Users

ALDSP Administration Console provides different privileges to different user entitlements. ALDSP now has the domain, admin, monitor, and browser entitlements. The domain level user is created by default and can assign entitlements to a user. The user privileges within ALDSP Administration Console depend on the entitlements. For example, the monitor or browser entitlements can only view the configuration in the ALDSP Administration Console, whereas the admin entitlement allows a user to change the configuration.

For more information, see Chapter 5, "Securing ALDSP Resources."

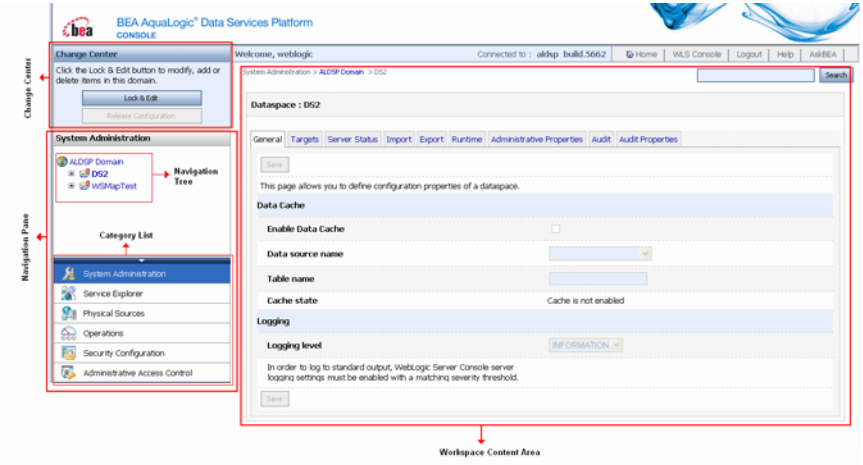
Introducing the ALDSP Administration Console

The ALDSP Administration Console is a web-based user interface to configure and administer ALDSP runtime server or cluster. You can use the ALDSP Administration Console to set security and caching policies for data services and configure ALDSP runtime settings such as thread usage and logging levels. In addition, you can deploy, import, and export dataspaces using the console and view metadata that is required by both developers and administrators.

Note: For more information, see Chapter 7, "Viewing Metadata Using the Service Explorer."

Figure 1-2 shows the main page of the ALDSP Administration Console.

Figure 1-2 ALDSP Administration Console



ALDSP Administration Console Components

The DSP Console constitutes the Navigation Pane and the Workspace Content area as shown in Figure 1-2. The navigation pane consists of the change center, navigation tree, and the category-based tabs. You can use this pane to access the deployed dataspace, functions, and web services. In addition, you can view and manage data in different categories such as the physical data sources and administrative access control.

Table 1-1 briefly describes the functions of each component in ALDSP Administration Console:

Table 1-1 Functions of ALDSP Administration Console Components

Component	Usage
Change Center	The change center is used to acquire and release a lock for editing the configuration within the console in a transactional manner. For more information, refer to "Change Center and Configuration Locking" on page 2-7
Navigation Tree	The navigation tree shows the artifacts stored on the server. The artifacts displayed in the workspace content area depend on the category you select from the list of category-based tabs. The navigation tree is rooted to the ALDSP domain. For more information, refer to "Navigation Tree and Category List" on page 2-10.

Table 1-1 Functions of ALDSP Administration Console Components (Continued)

Component	Usage
Category List	The category-based tabs or the category list provides specific information about the deployed dataspace, web services, and functions. Each tab in the list provides a set of artifacts for the selected project, data service, or function. For more information, refer to "Navigation Tree and Category List" on page 2-10.
Workspace Content Area	The workspace content area displays the artifacts based on the selection in the navigation tree and the category list. It allows you to configure system administration tasks, import, export, and deploy dataspaces, work with security configurations, manage data caching, and auditing tasks. For more information, refer to "Using the Workspace Content Area" on page 2-11.

Server Classpath Settings

The following table provides classpath requirements for servers running ALDSP.

Listing 1-1 Server Classpath Settings for ALDSP-Enabled Servers

```
CLASSPATH=
   <ALDSP_HOME>/lib/ld-server-core.jar
   <WLS_CLASSPATH>
```

Notes:

- ALDSP depends on the apache_xbean.jar, which is available from the weblogic.jar manifest entry.
- The ld-server-core.jar's manifest file refers to the following set of jars used by the ALDSP server.

```
sdo.jar
wlsdo.jar
binxml.jar
xquery.jar
../external/com.bea.common.configfwk.wlinterop_9.2.2.0.jar
../external/com.bea.common.configfwk_1.0.0.0.jar
../external/alsb_client_9.2.jar
../external/jgrapht-jdk1.5.jar
relational-providers.jar
ld-client.jar
```

CHAPTER 2

Getting Started with ALDSP Administration

With ALDSP 3.0 release, you need to create and configure WebLogic Server 9.2.x unlike the previous ALDSP releases, which used WebLogic Server 8.1. Before you start working with ALDSP development environment, you need to deploy your dataspace project on a WebLogic domain enabled for ALDSP. Using WebLogic Server 9.2.x, you can create users and groups for ALDSP and manage their permissions.

Most of the other administrations tasks for ALDSP 3.0, can be performed through the ALDSP Administration Console and therefore you may not need to launch the WLS Administration console frequently. Table 2-1 lists the tasks that you can perform using ALDSP Administration Console and the ones that you need to perform using WebLogic Server Administration console.

Table 2-1 Administration Tasks for ALDSP Administration Console and WLS Administration Console

Task	Administered Through
ALDSP Users and Groups: Chapter 5, "Securing ALDSP Resources" Also refer to WebLogic Server user and groups.	WebLogic Server Administration Console
Deployment: Chapter 3, "Deploying Dataspaces"	ALDSP Administration Console
Security: Chapter 5, "Securing ALDSP Resources"	ALDSP Administration Console
Caching: Chapter 8, "Configuring Query Results Cache"	ALDSP Administration Console
Auditing: Chapter 9, "Working With Audit and Log Information"	ALDSP Administration Console

For more information about creating and configuring a new server for ALDSP, refer to Post-Installation Tasks in AquaLogic Data Services Platform *Installation Guide*:

This chapter describes the tasks that you can perform using ALDSP Console and also provides steps to start and stop the WebLogic Server. It contains the following sections:

- Updating the ALDSP License
- Starting and Stopping WebLogic Server
- Launching ALDSP Administration Console
- Exploring ALDSP Administration Console

Updating the ALDSP License

ALDSP requires a valid product license to run. The ALDSP license is included as a component in the WebLogic Server license file, `license.bea`. To apply or update an ALDSP license file, use the BEA UpdateLicense utility to update the `license.bea` file.

For details about BEA product licensing, see Installing and Updating WebLogic Platform License Files in *Installing WebLogic Platform* of the WebLogic Server documentation.

Starting and Stopping WebLogic Server

To start working with the ALDSP development environment and to administer the WLS enabled for ALDSP, you must first start WebLogic Server. Although you may not need to stop WebLogic Server frequently, it may be required in certain situations. This section describes how to start and stop WebLogic Server (WLS) in a standalone WebLogic domain, after you have configured your WebLogic Server 9.2.x.

Note: If you are already running an instance of WebLogic Server that uses the same listener port as the one to be used by the server you are starting, you must stop the first server before starting the second server.

Starting the Server

1. At the command prompt, navigate to the domain directory.

   The domain directory is *BEA_HOME*/user_projects/*domain_name*. An example could be c:\bea\user_projects\domains\mydomain.

2. Run the server startup script: `startWebLogic.cmd` (Windows) or `startWebLogic.sh` (UNIX).

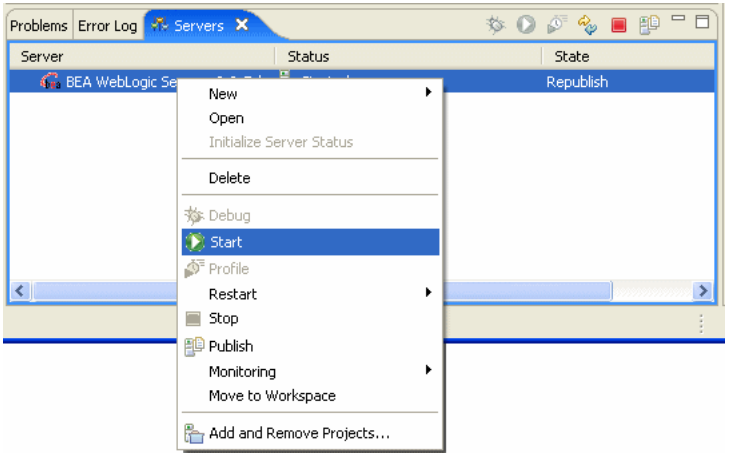
   The startup script displays a series of messages, finally displaying a message similar to the following:

   ```
   <Dec 8, 2004 3:50:42 PM PDT> <Notice> <WebLogicServer> <000360> <Server
   started in RUNNING mode>
   ```

You can also start WebLogic Server through the eclipse-based IDE for ALDSP. To start the server:

1. Open the IDE and click the Servers tab.
2. Right-click the server that you have configured and select Start, as shown in Figure 2-1. If you want run the server in debug-mode then select Debug. This starts WebLogic Server.

Figure 2-1 ALDSP IDE: Starting WebLogic Server



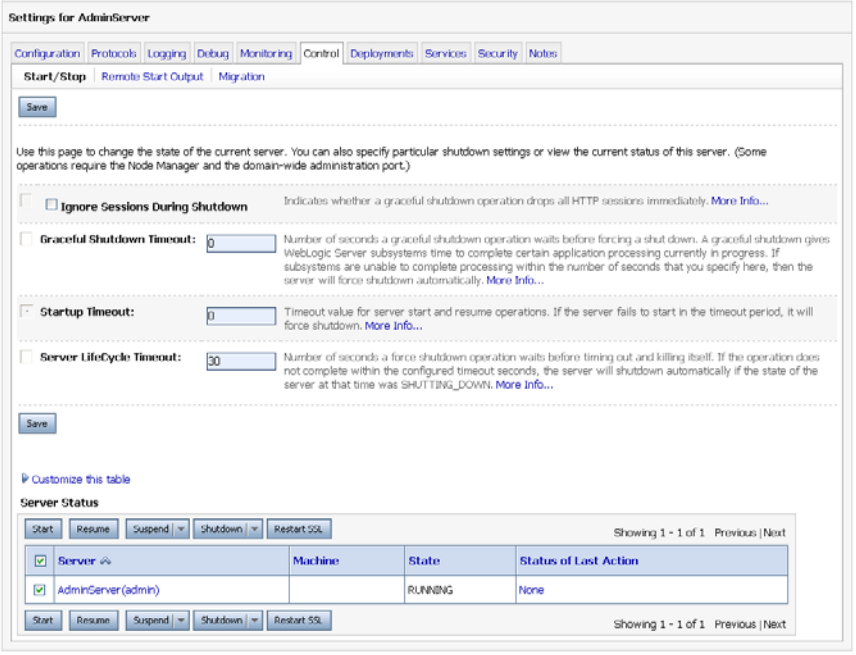
Stopping the Server

To stop the WebLogic Server using the eclipse-based IDE, right-click the server listed in the Servers tab, as shown in Figure 2-1 and select **Stop**.

Alternatively, you can stop a WebLogic Server instance that is running a dataspace project from the WebLogic Administration Console.

1. Start the WebLogic Server Administration Console.
2. Acquire the lock by clicking Lock & Edit.
3. In the left pane, click to expand Environment and select Servers.
4. Select the server instance you need to stop.
5. Click the Control tab. The Start/Stop tab is displayed, as illustrated in Figure 2-2.

Figure 2-2 Graceful Shutdown of a Server



6. Specify the graceful shutdown timeout limit incase you need to do a force shutdown after some time.
7. From Server Status table, click the Shutdown list.
8. Select the When work completes option.
9. Select Yes to confirm shutdown. This shuts down the selected server after all the pending tasks are completed.

Launching ALDSP Administration Console

The AquaLogic Data Services Platform Administration Console is a web-based interface that enables you to administer and manage dataspace projects, access metadata, and configure security and caching policies.

Before you launch the ALDSP Administration Console, make sure that the WebLogic Server is started. For more information about starting WebLogic Server, see "Starting the Server" on page 2-3. To launch ALDSP Administration Console:

1. Open the following URL:

   `http://hostname:port/dspconsole`

   Where:

   – *`hostname`* is the machine name or IP address of the host server

   – *`port`* is the address of the port on which the host server is listening for requests (7001 by default)

   For example, to start the ALDSP Administration Console on a local instance of WebLogic Server (running on your computer), navigate to the following URL:

   `http://localhost:7001/dspconsole/`

2. When the login page appears, enter the appropriate user name and password.

   The default user name and password is weblogic/weblogic, respectively.

Note: The discussion and examples in the following chapters of this book (*Administration Guide*) assume that you have:

- Installed the current version of ALDSP.
- Build at least one dataspace as described in the *Data Services Developer's Guide*. Building a dataspace automatically deploys it and any data services it contains on your currently running WebLogic Server.

In case you need to launch the WLS Administration console, click the WLS Console link on the top-right corner of ALDSP Administration Console, as shown in Figure 2-3.

Figure 2-3 WLS Console Link in ALDSP Console

For more information about starting the WebLogic Administration Console, refer to Starting the Administration Console section in *Introduction to WebLogic Server and WebLogic Express*.

http://edocs.bea.com/wls/docs91/intro/console.html#1122070

Exploring ALDSP Administration Console

This section provides details about using different components of the ALDSP administration console. It includes the following topics:

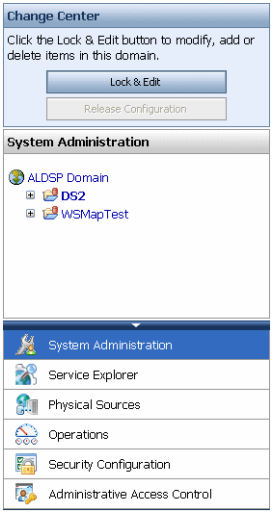
- Using the Navigation Pane
- Using the Workspace Content Area

Using the Navigation Pane

You can use the navigation pane to view the navigation tree and all the data services, functions, and web services. The change center allows you lock and edit the configuration settings within the console and then save or discard changes depending on your requirement. Using the category-based tabs from the category list, you can view and manage the artifacts related to each tab, including the system administration tasks such as deployment of data services, importing and exporting data service JAR files, and auditing.You can also view metadata, manage caching, and configure security settings using the category-list.

Figure 2-4 displays the components of the navigation pane.

Figure 2-4 Navigation Pane



This section describes the functions of some of the components of the navigation pane in detail.

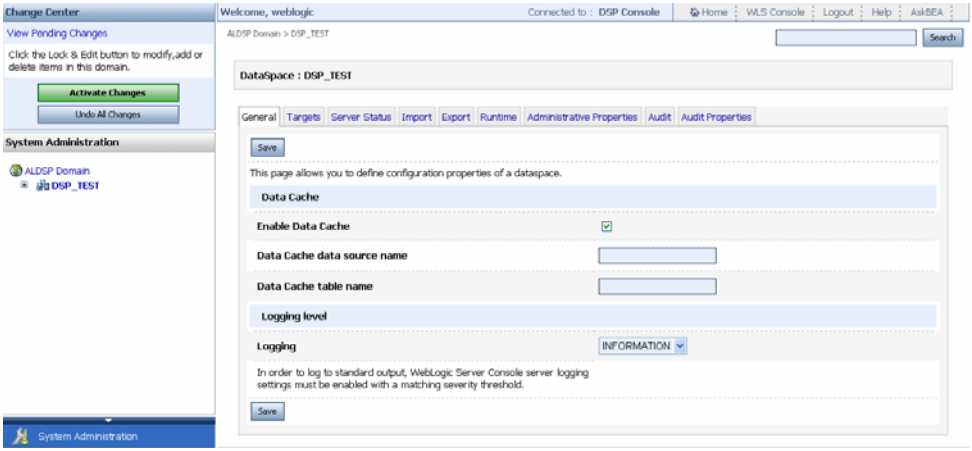
Change Center and Configuration Locking

The change center feature in ALDSP Administration console is similar to the WLS Administration console. It enables you to acquire a global lock over the console configuration, make one or more changes to the configuration, if required, and then activate or discard the changes. The configuration settings are edited in a transactional manner, therefore, only one user can acquire the lock to the console.

To acquire the lock and then activate or discard changes:

1. Click Lock & Edit option from the change center. This enables you to make changes to the workspace.
2. Save the changes in the Workspace Content Area by clicking **Save**. The message “Settings updated successfully” is displayed in the workspace content area.
3. From the change center area, click **Activate Changes** or **Undo All Changes**, as shown in Figure 2-5, to activate or discard the changes. If you click Activate Changes, then the message “Changes activated successfully“ is displayed in the workspace content area and if you select Undo All Changes, then the “Changes discarded successfully” message is displayed.

Figure 2-5 Activating/Deactivating Configuration Changes



The change center feature is available only to the domain and admin entitlements for a resource configured for security in ALDSP. Other ALDSP entitlements cannot use the change center. For more information about user entitlements, refer to the *Administrative Access Control* section in Chapter 5, "Securing ALDSP Resources."

You do not need to acquire a lock to edit the configuration within the administration console in the following cases:

- To create and delete dataspaces, you do not need to explicitly acquire a lock because the system acquires the lock by default. For more information about creating and deleting dataspaces, refer to Chapter 3, "Deploying Dataspaces."
- Security policies, in both runtime security and administrative access control categories, do not require the change center lock. The policies are stored in a separate repository, in WLS configuration, and therefore do not take part in the ALDSP configuration session. For more information, refer to Chapter 5, "Securing ALDSP Resources."

Based on the operations performed using the change center, the change center behavior may differ. Table 2-2 lists and describes the change center behavior in different situations:

Table 2-2 Change Center Behavior

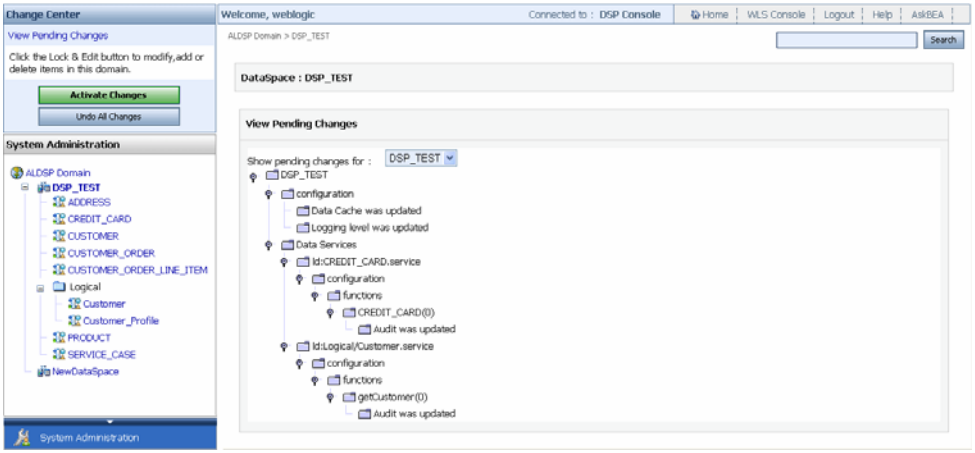
Condition	Behavior
User does not have domain or admin entitlements for any of the ALDSP resources such as a dataspace or data service.	User is denied access and the change center is disabled.
Lock has not been acquired by any one and can be acquired by the logged in user.	The user can acquire the lock to the change center and perform configuration changes.
Lock has been acquired by the logged in user and changes are made.	The change center provides the option to activate or discard changes. So, the Activate Changes and Undo All Changes options appear in the change center area.
Lock has been acquired by some other user but the logged in user being a domain user is allowed to forcibly acquire the lock.	The change center displays the Take Lock && Edit option if the user has domain entitlements for the dataspace.

For more information about using the change center, you can also refer to: http://edocs.bea.com/wls/docs92/intro/console.html#wp1122447

Pending Changelist

The pending change list displays the difference between the current session values and the core values. The dataspace artifacts that are created, updated, or deleted are displayed in the pending changelist. Pending changes are shown in the tree view, as shown in Figure 2-6, whereas the configuration changes are shown in the leaf node.

Figure 2-6 Pending Changelist



Navigation Tree and Category List

There are six categories in the ALDSP administration console. The artifacts displayed in the workspace content area for a data service, function, or web service depend on the category-based tab that you select from the category list.

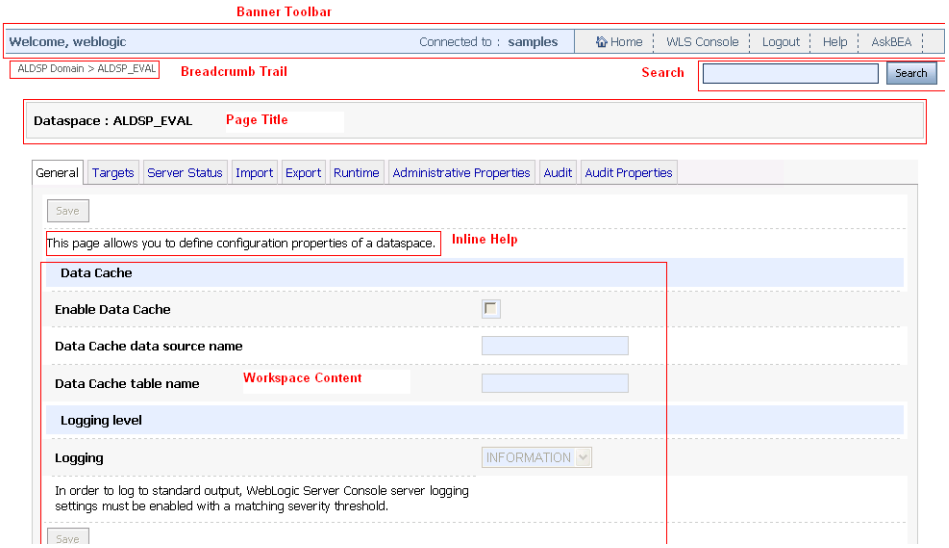
The following list describes the function of each category tab:

- **System Administration:** This is the default tab that is displayed when you log into ALDSP console. It provides functionality to set the state and the target server for deployment, importing and exporting of project JAR files, checking the administrative properties, and auditing.
- **Service Explorer:** The service explorer provides metadata artifacts for the deployed dataspace project, function, and web services including native web services. For more information, refer to Chapter 7, "Viewing Metadata Using the Service Explorer."
- **Physical Sources:** This tab provides details about the different physical data sources that are deployed on the server. The physical data sources can include delimited files, java functions, relational databases, web services, and XML files.
- **Operations:** This tab allows you monitor the active queries, data cache size, and active updates for a dataspace.
- **Security Configuration:** This tab allows you set runtime security policies for securable resources such as dataspaces, data services, functions, and web services. For more information, refer to "Understanding Runtime Security Policies" section in Chapter 5, "Securing ALDSP Resources."
- **Administrative Access Control:** This tab enables you set the administrative access control policies for different users who need to access ALDSP Administration Console. For more information, refer to the "Working with Administrative Access Control Policies" section in Chapter 5, "Securing ALDSP Resources.":

Using the Workspace Content Area

The workspace content area displays artifacts based on the tab selected in the category list and the node selected from the navigation tree. It consists of various options that enable you to view, search, configure, and audit ALDSP resources. Figure 2-7 displays the workspace content area that is displayed when you log in to the console.

Figure 2-7 Workspace Content Area



As illustrated in this figure, the workspace content area constitutes the following:

- **Banner Toolbar:** It shows the user name and the server that you are logged into. The links on the right, allow you to log into the WLS Console, logout of ALDSP Administration Console, along with help options.
- **Breadcrumb Trail:** It displays the current category and the resource that you select from the navigation tree. You can access the category or resource using the trail links also.
- **Search:** This field is you to search metadata. When you click Search, the system starts a search across all artifacts on the server and displays the results in a search result page. If you click Search without entering any value in the field, the Advanced Search page is displayed. For more information, refer to "Searching Metadata" on page 7-16.

- **Page Title:** This displays the current artifact that you access on the ALDSP Administration Console.
- **Inline Help:** This help is available on each page of the console and provides guidance about using the options on the console.
- **Workspace Content:** This area displays information about the resource depending on the category you select from the category-list.

CHAPTER 3

Deploying Dataspaces

This chapter describes how to deploy dataspaces to an Administration Server, a Managed Server, and a cluster. It also describes how to migrate dataspaces from development to production.

The chapter contains the following sections:

Introduction

ALDSP Administration Console provides you the ability to deploy, export and import dataspaces. Using the console, you can export, import, and delete dataspaces that are deployed on a WebLogic Server without interrupting other running dataspaces. In addition, you can import artifacts to an existing dataspace without interrupting existing requests running against that dataspace

During development, you can deploy dataspaces to a WebLogic Server directly from the eclipse-based IDE. After development, you can deploy dataspaces to production WebLogic Servers using the ALDSP Administration Console or the IDE.

Creating a New Dataspace

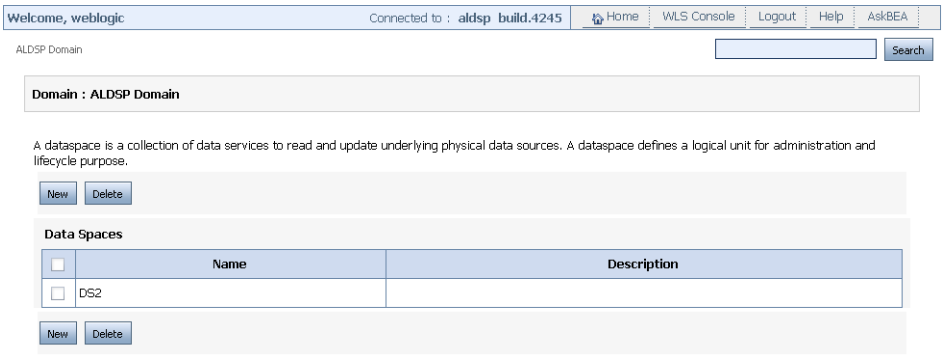
You can create a new dataspace using the ALDSP Administration Console and associate a JAR file with it. This enables you to create and manage the dataspace on the server directly.

Note: Only a domain user has the ability to create a new dataspace. For more information about domain users, refer to Working with Administrative Access Control Policies section in Chapter 5, "Securing ALDSP Resources."

To create a new dataspace in the ALDSP-enabled WebLogic domain:

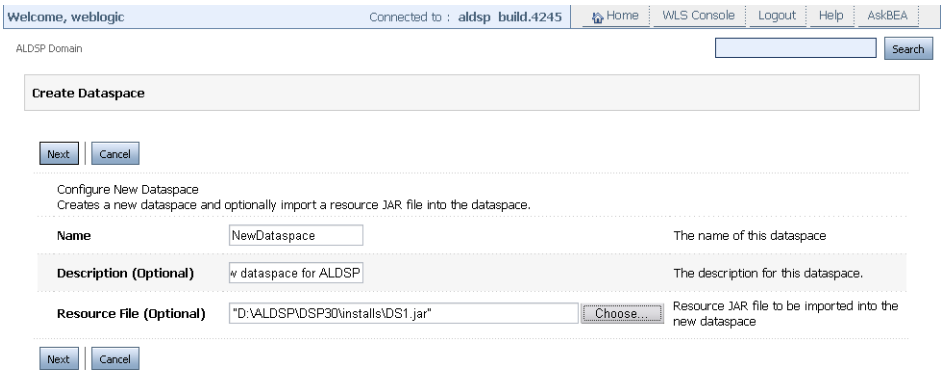
1. Click the System Administration category from the navigation pane.
2. Select the domain node.
3. From the workspace content area, click New as shown in Figure 3-1.

Figure 3-1 Creating a New Dataspace



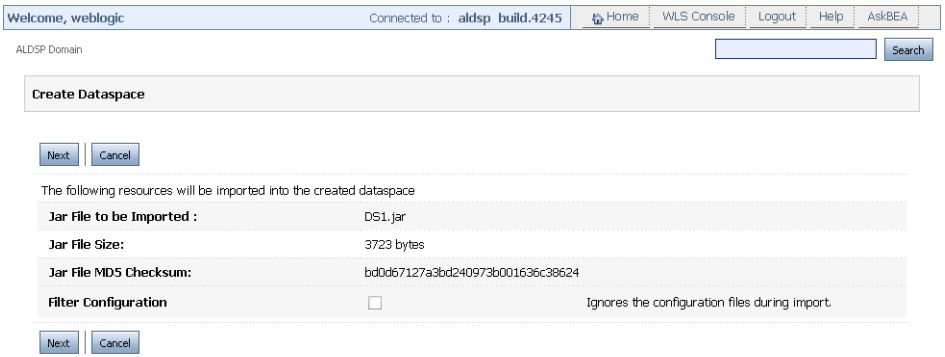
4. On the Create Dataspace page shown in Figure 3-2, specify the following:
   - Name: Name of the new dataspace that you want to create.
   - Description: An optional description of the dataspace.
   - Resource File: A JAR file that you want to import in the dataspace. This is optional.

Figure 3-2 Specifying the New Dataspace Details



5. Click Next. This displays details such as the file size and checksum information about the resource file being imported as shown in Figure 3-3.

Figure 3-3 Resource File Details



6. On this page, select the Filter Configuration checkbox if you do not want to import the resource file configuration. To retain the resource file configurations, make sure that you do not select the Filter Configuration checkbox.

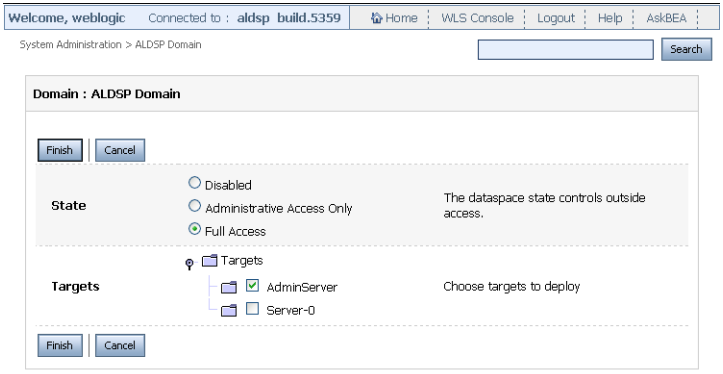
   Each dataspace contains one .space file that contains all the global dataspace properties. For example, for a dataspace my_dspace_DS there is a corresponding file named My_DSpace.space. The dataspace also contains one file, named My_DSpace.sources, that contains all the properties pertaining the physical sources used by the dataspace My_DSpace.

   For each dataservice (.ds) file contained in the dataspace, there is a .service file named after the dataservice and located within the same folder as the data service, that carries the data service configuration properties.

   Finally, a dataspace may contain one or more .xml files under the folder `DSP-INF/service-accounts`, which carry service account information details.

7. Click Next. This displays the page where you can select the state and targets for the dataspace as shown in Figure 3-4.

Figure 3-4 Selecting the State and Target Server for a Dataspace



A deployed dataspace can be in one of the following states:

- Disabled: The dataspace is not live and cannot be administered from the console.
- Administrative Access Only: The dataspace is accessible only to the Administrator.
- Full Access: This dataspace is accessible to all authorized users.

8. Specify the state and target server and click Finish to create and deploy the new dataspace.

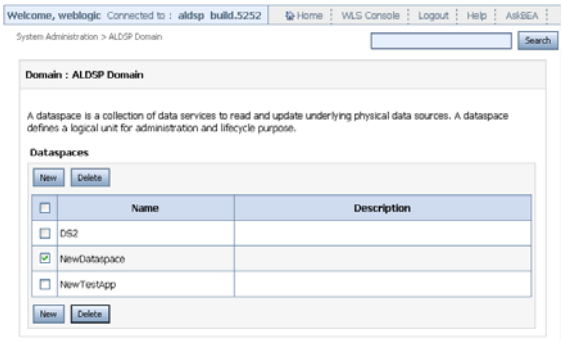
Note: You may need to wait for sometime before the new dataspace is deployed successfully depending on the size of the dataspace.

Deleting a Dataspace

Only a domain user can delete a deployed dataspace. To delete a dataspace:

1. Navigate to the ALDSP Domain level.
2. Select the dataspace that you need to delete as shown in Figure 3-5.

Figure 3-5 Selecting the Dataspace to Delete



3. Click Delete. The next page confirms if you want to delete the dataspace. Select Yes to delete the dataspace.

Note: If you delete the target Managed Server on which your dataspace is deployed, the dataspace deletion will fail.

Deploying Dataspaces on a Target Server

Deployment is done through the System Administration category in the ALDSP Administration Console. ALDSP dataspaces can only run in an ALDSP-enabled WebLogic domain. You can create a new WebLogic domain using the Configuration Wizard.

Note: For more information about using the Configuration Wizard to set up an ALDSP-enabled WebLogic domain, refer to Creating a New Domain section in Chapter 1, "Overview of ALDSP Administration."

The Configuration Wizard automatically transfers the required items to the target server. These include the ALDSP dataspace artifacts, with the corresponding configuration and binary files, as well as WebLogic components such as data source connections and pools. When you move a dataspace from the development to production, you need to make sure that these items are transferred to the target production server.

Note: A target server can be an Administration Server, a Managed Server, or a cluster. The steps to deploy dataspaces on any of these targets are the same.

An Administration Server is the central configuration repository for the set of WebLogic Servers in a domain.

You can deploy a dataspace on multiple Managed Servers and clusters depending on your requirement. To deploy dataspace artifacts on a Managed Server or a cluster, you must first create a Managed Server or cluster using the Configuration Wizard.

Note: If you need to deploy a Web Service Map on a cluster, then you need to specify the cluster address. For details, refer to Deploying a Web Service Map on a Cluster.

For more information about creating Managed Servers, refer to the Create Managed Servers topic in *WebLogic Server Administration Console Online Help*.

For more information about creating clusters, refer to the Create a Cluster topic in *WebLogic Server Administration Console Online Help*.

Deploying a Dataspace

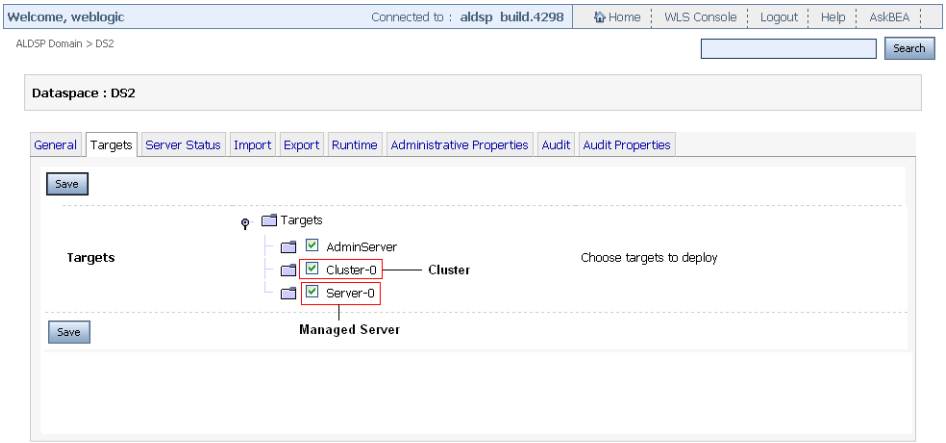
To deploy a dataspace on WebLogic Server using ALDSP Administration Console:

1. Start the ALDSP Administration Console.

   For more information, see Launching ALDSP Administration Console section in Chapter 2, "Getting Started with ALDSP Administration."

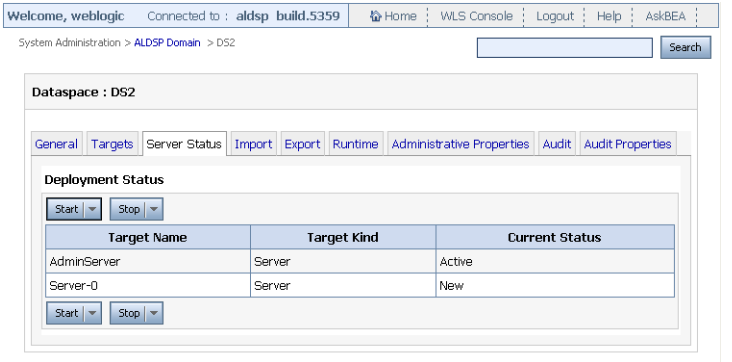
2. Select the System Administration category and then select the Targets tab from the workspace content area as shown in Figure 3-6.

Figure 3-6 Deploying a Dataspace on a Target Server



3. Select the target server on which you want to deploy the dataspace.
4. Click Save. This set the target server for your ALDSP dataspace.
5. To view or change the deployment status, click the Server Status tab as shown in Figure 3-7.

Figure 3-7 Checking the Server Status



The options to start and stop the target servers are mentioned in the following table:

Table 3-1 Options for Starting and Stopping Server

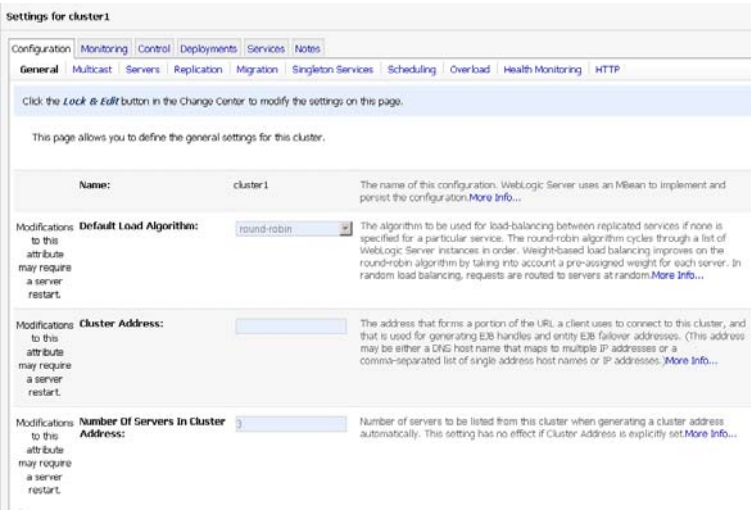
State	Option	Description
Start	Servicing Administration Requests	Select this option when the dataspace is accessible only at the Administration level. This usually happens when the dataspace project is deployed on the production server and is yet to go live.
Start	Servicing All Requests	Select this option when the dataspace is ready to service all client requests.
Stop	Servicing Non-Administration Requests	Select this option when you need to stop servicing requests from clients but continue servicing administration requests.
Stop	Servicing All Requests	Select this option if you need to stop servicing requests from all clients including administration requests.

Deploying a Web Service Map on a Cluster

Before you deploy a web service map on a cluster you need to specify the cluster address using the WebLogic Server console. If you do not specify the cluster address then the WSDL creation for the web service map fails.

To specify the cluster address on WebLogic Server, specify the cluster address in the Configuration > General tab for the cluster as shown in Figure 3-8.

Figure 3-8 Specifying Cluster Address on WebLogic Server



For detailed information about configuring clusters on WebLogic Server, refer to Create and Configure Clusters.

Importing Dataspace Artifacts

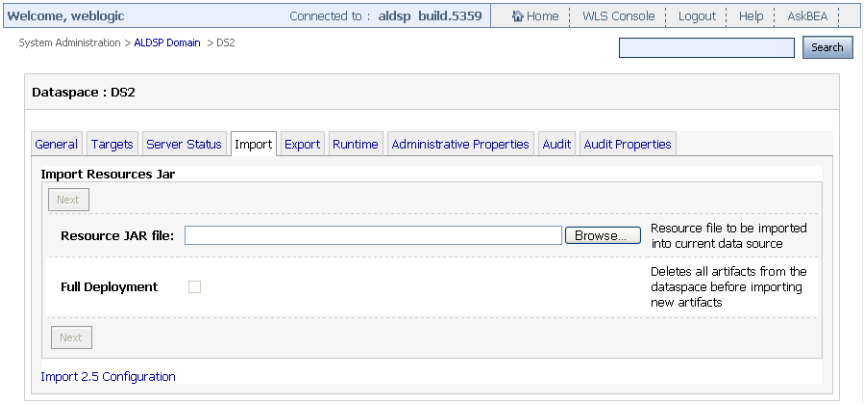
ALDSP Administration Console allows you to perform incremental and full deployment of resource JAR files. This section describes the steps to perform incremental and full deployment through the ALDSP Administration Console.

Note: You can also import the data service configuration settings from ALDSP 2.5. This enables you to use the same configurations that you used in the ALDSP 2.5 environment, while continuing to work with ALDSP 3.0. For information about migrating an ALDSP 2.5 application to 3.0, refer to Migrating from ALDSP 2.5 to ALDSP 3.0.

To perform incremental or full deployment of resource files:

1. Acquire the lock by selecting Lock & Edit.
2. From the Navigation pane, select the System Administration category and then select the dataspace in which you want import configuration or artifacts.
3. Click the Import tab as shown in Figure 3-9.

Figure 3-9 System Administration Category: Import Tab



4. Browse and specify the resource file path in the Resource JAR File box.
5. If you want to perform full deployment, then select the Full Deployment check box from the Import Resource Jar section. If you select this option, then the system deletes all the artifacts from the dataspace and then imports the new artifacts.

6. If you want perform incremental deployment, then do not select the Full Deployment check box. In case of incremental deployment, ALDSP updates only those dataspace artifacts that have changed and add any new artifacts.

7. Click Next to move to the page that displays the resource JAR file details, which include file checksum details and file size as shown in Figure 3-10. In addition, this page provides the following options:

   a. Filter Configuration: Select this option if you do not want to import the configurations of the resource file.

   b. Preserve End Point Mappings: Select this option if you want import all the configuration and the resources (artifacts) but keep the old endpoint mappings intact.

      This option is useful when you move configurations from the staging server to a live production server. On the staging server, you configure and test the configurations. If the testing is successful, move the configurations from the staging to the production server. However, the endpoints used during staging and production would not be the same as you would not be testing directly on production server. So, when you import mappings from the staging server, you may want to retain the mappings that already exist in the production database. In that case, select the Preserve Endpoint checkbox.

Figure 3-10 Resource File Details

8. After selecting options on this page, click Import. When the import is completed, the message "Import operation was successful" is displayed.

Note: Depending on the size of the files and the topology of your domain, the import operation may take time, therefore you may need to wait for import to complete.

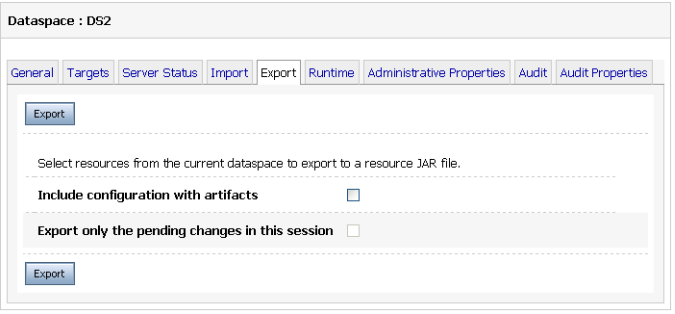
9. Click Activate Changes from the change center to activate the import.

Exporting Dataspace Artifacts

You can export dataspace artifacts with or without retaining the configuration settings. To export dataspace artifacts:

1. Click the System Administration Category and the dataspace that you want to export as a JAR.
2. Click the Export tab.
3. Select the Include configuration artifacts check box as shown in Figure 3-11, if you want to export the configuration along with all the artifacts.

Figure 3-11 Export Tab



4. If you are already in a session and want to export changes that have occurred within that session then select the Export only the pending changes in this session check box.

Note: The Export only the changes in this session check box is enabled only when the lock is acquired.

5. Click Export
6. Specify the location where you want to save the dataspace artifacts and the file is saved as a JAR file at the specified location.

CHAPTER 4

Configuring ALDSP Resources

This chapter describes how to configure an ALDSP dataspace including tasks such as creating administrative properties, managing memory, and enabling cache. It contains the following sections:

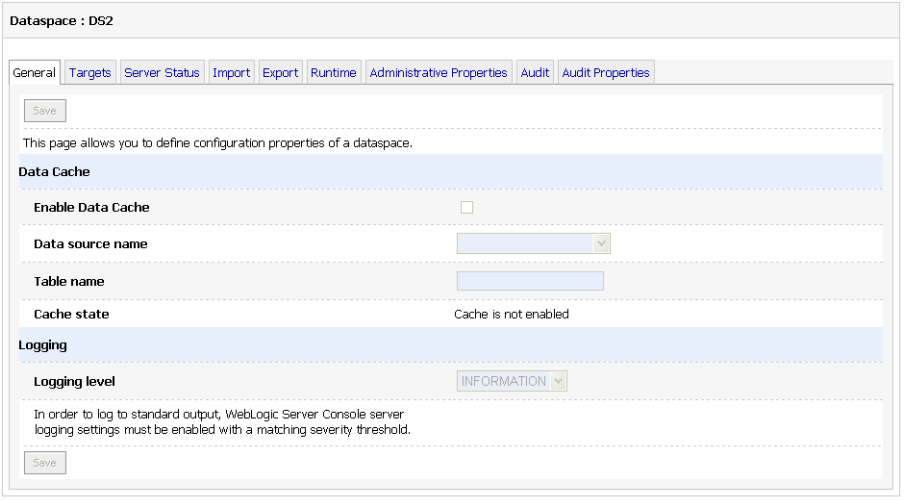
Configuring the Cache and Log for a Dataspace

You can view and configure settings for a dataspace such as caching and logging using the General tab in the System Administration category.

To configure general dataspace settings:

1. Select the System Administration category and then the dataspace from the navigation tree. The General tab appears as shown in Figure 4-1.

Figure 4-1 General Dataspace Settings Page



2. Acquire the lock to make changes to the general configuration of the dataspace.
3. You can enable data caching and logging level details using this page. For more information on data caching, refer to Chapter 8, "Configuring Query Results Cache." For more information on logging, refer to Chapter 9, "Working With Audit and Log Information."
4. Click Save > Activate Changes.

Using the Physical Sources Category

The Physical Sources category allows you to configure and modify the resource end points, view the location of physical data sources, and create substitute SQL statements.

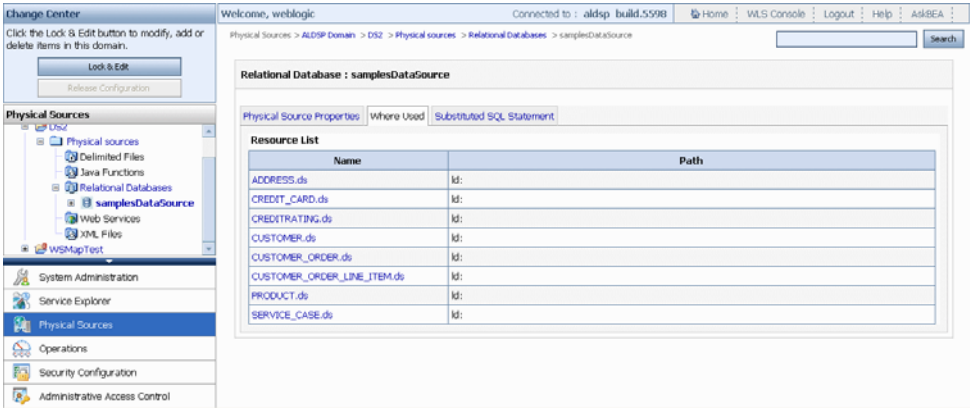
This section provides details about configuring each of these features using the Physical Sources category on the ALDSP Administration Console. It includes the following topics:

- Viewing Physical Data Source Locations
- Modifying Data Source End Points
- Substituting SQL Statements

Viewing Physical Data Source Locations

You can view a list of data services and function libraries that use the defined relational databases. Click the **Where Used** tab to view the list of data services and the corresponding paths (Figure 4-2).

Figure 4-2 Physical Data Services Relational Dependencies

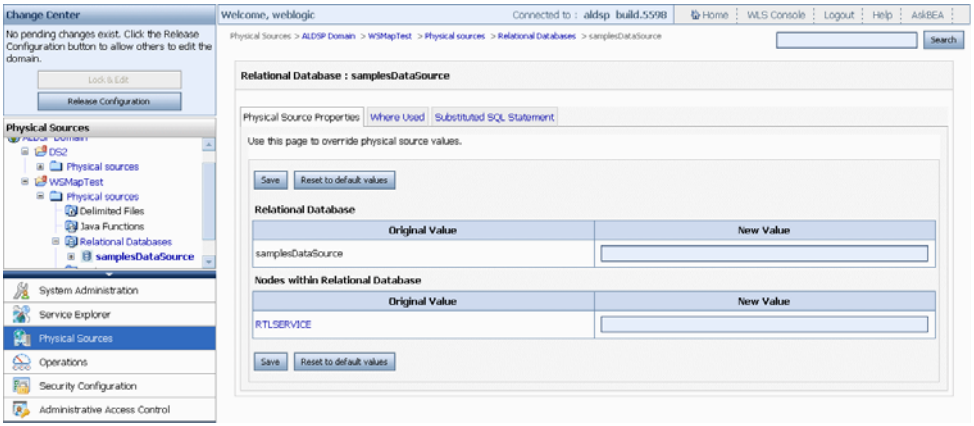


You can select a data service from the Resource List to view the metadata about the data service.

Modifying Data Source End Points

When you move dataspaces from development to production server, you may need to change the location of data sources or names of other artifacts. For example, if you are using sample data sources during development to protect confidential or otherwise secured information, you need to substitute a new data source with the actual data for the test version. You can make these changes through the Physical Sources category as shown in Figure 4-3.

Figure 4-3 Setting End Points for Relational Sources



By modifying the data source endpoints, you can change the name and location of a data source as well as the target names of subordinate artifacts. In the case of relational sources this includes names of catalogs, schemas, packages, tables, stored procedures, views, and relational functions. End point modifications are effective until they are further modified or reverted to the original value.

To reset the original value to the end point name:

1. Acquire the lock by clicking Lock & Edit.
2. Click **Reset to original value**. This option will not revert the value to the previous setting, instead it will directly revert it to the original name. If you assign some intermediate target names and click **Reset to original value**, the values revert to the same values as those in the **Original Value** column.
3. Click Save > Activate Changes.

Note: If you change the end point for an artifact, some of the properties for the artifact should match with the old source. For example, the Vendor type and version properties for a relational data source should be identical with the old source.

Table 4-1 identifies the artifacts whose end point settings can be changed.

Table 4-1 Artifacts for which End Points can be Modified Through the ALDSP Administration Console

Data Source Type	Artifact
Relational	Data source name and location
	Catalog
	Schema
	Package
	Table
	Views
	Relational functions
	Stored procedure
Web Service	Web service name and location
	Service
	Port
	Operation
XML Content	Data source name and location
Delimited File Content	Data source name and location

Substituting SQL Statements

ALDSP uses SQL to access relational data sources. At compilation time, the built-in query optimizer determines the best execution strategy for backend sources. Then SQL queries are generated and submitted to underlying databases.

SQL queries generated by the relational wrapper are specific to each underlying database. While the SQL queries that are generated typically produce good results, there are cases when further optimization of the generated queries is desirable. In most RDBMS systems, such optimization is done through execution *hints*.

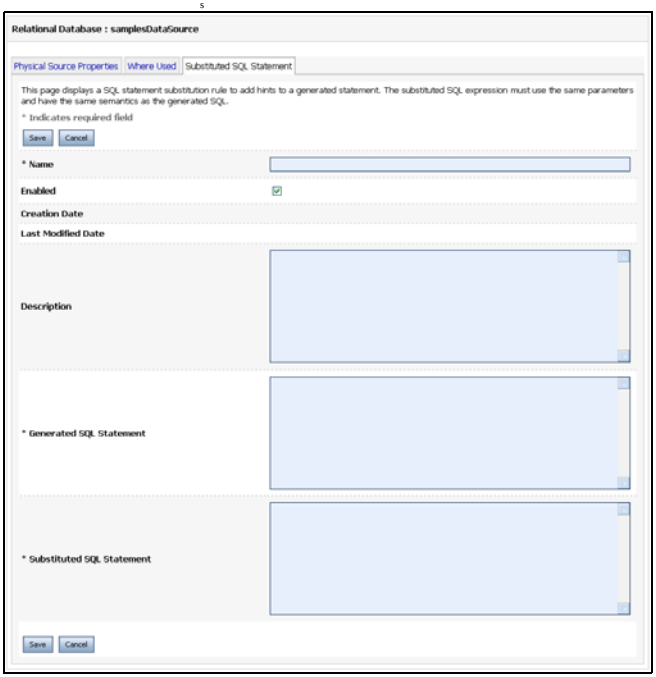
SQL statement substitution allows you to add hints to generated SQL queries by providing edited SQL statements that will be executed instead of the query that is generated by ALDSP by default.

WARNING: Unlike SQL statements generated by ALDSP, substituted SQL statements are passed to the underlying database without validation. For this reason, users are strongly advised against using this feature for any purpose other than providing hints to the database. It is also recommended that prior to deployment any substituted SQL statement be tested against its generated counterpart to make sure that the expected performance advantage is obtained.

Substitute SQL statements are created and registered in the ALDSP Administration Console using the Substituted SQL Statements tab available through the Physical Sources category as shown in Figure 4-4.

Figure 4-4 Substituted SQL Statement Dialog Box

s



How SQL Statement Substitution Works

ALDSP server maintains a substitution table between the original generated SQL queries and any replacement queries supplied by the user. Only SQL queries specified by user will be substituted.

The ALDSP administrator defines and maintains substitution queries through the ALDSP Administration Console.

The replacement query is executed instead of the original SQL query. The ALDSP runtime engine reads the SQL result set using type/column information of the original query. Potential problems related to incorrect substitution, which violates the conditions listed in Requirements for SQL Statement Substitution include the following problems:

- Incorrect result returned by XQuery, for example, incorrect data, no result at all, incorrect order of the result, are among the possible unwanted outcomes.
- Error generated by the runtime engine during SQL statements execution, for example, problems with parameter binding and reading the result.

Supporting Externalized End Points in Substituted Queries

In both the generated and substitute queries, a special syntax is used to support externalized end points (see "Modifying Data Source End Points" on page 4-5 for details). The following substituted queries show this syntax (emphasis added):

```
SELECT /*+ FIRST_ROWS (10)*/ t1."BILL_TO_ID" AS c1, t1."C_ID" AS c2,
t1."DATE_INT" AS c3, t1."ESTIMATED_SHIP_DT" AS c4,
t1."HANDLING_CHRG_AMT" AS c5, t1."ORDER_DT" AS c6, t1."ORDER_ID" AS c7,
t1."SALE_TAX_AMT" AS c8,
t1."SHIP_METHOD_DSC" AS c9, t1."SHIP_TO_ID" AS c10, t1."SHIP_TO_NM" AS
c11, t1."STATUS" AS c12,
t1."SUBTOTAL_AMT" AS c13, t1."TOTAL_ORDER_AMT" AS c14, t1."TRACKING_NO"
AS c15
FROM {RTLAPPLOMS}.{CUSTOMER_ORDER} t1
```

Note: If you are adding SQL fragments (such as string literals) in your substituted SQL statement, you also need to use the convention of doubling opening curlie braces.

For example:

```
SELECT t1.ID FROM CUSTOMER() WHERE $i/ID > 'a{bee}c' return $i/ID
```

is translated to:

```
SELECT t1.ID FROM {CUSTOMER} t1 WHERE t1.ID > 'a{{bee}c'
```

Depending on your requirement, specify replacement queries using the same name placeholders as the original query. At the end of the SQL generation stage the original names are replaced with the current end-point names. The original names are used if no end-point setting is found.

Requirements for SQL Statement Substitution

There are several requirements regarding the substituted SQL query:

- The query must return same data, with same number of columns and column types.
- Columns must be listed in the same order as the original query.
- The query must have the same number of parameters, in the same order, as the original query.
- The expected parameter types must match that of the original query.

- Alias column names must be exactly the same as in the original query.

  Note: For queries using sub-queries, the column aliases need to be preserved by only the outermost subquery and not the inner subqueries.

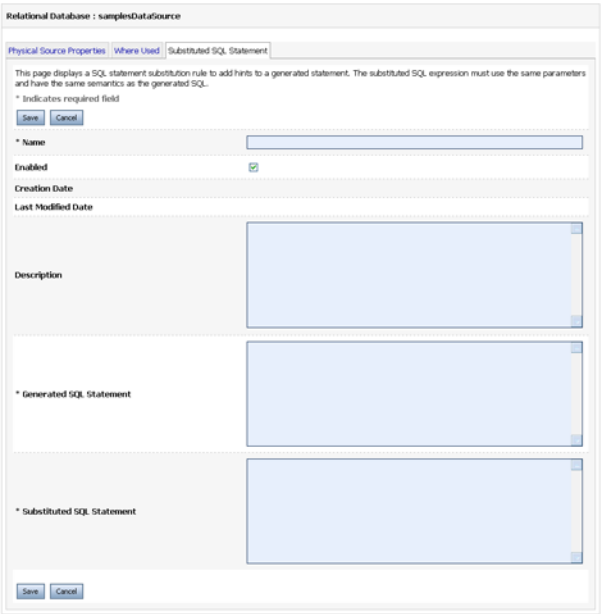
- If the original query contained an ORDER BY clause, the same ordering result must be required.

Creating Substitute SQL Query Statements

To create a substitute SQL query:

1. Click Lock & Edit to acquire the lock.
2. Select the Physical Sources category from the category list and then select the relational databases option from the navigation pane.
3. Navigate and select the relational data source for which you want to create the substitute query and then select the Substituted SQL Statement tab.
4. Click New. This displays the page where you can specify the SQL statement substitution rule as shown in Figure 4-5.

Figure 4-5 Rules for SQL Statement Substitution



5. Specify the following details on this page:

- Name of the substitute query
- Enable the substitute query
- An optional description of the query
- The SQL statement generated by ALDSP
- The substituted SQL statement

The system automatically tracks creation and last modified dates. An example for using the substitute query is available at SQL Statement Substitution Example.

SQL Statement Substitution Example

The order in which SQL statement substitutions are established is not fixed. Therefore, the example in this section and the steps involved are only one approach to creating and testing SQL statement substitution.

1. Setup your environment with these actions:

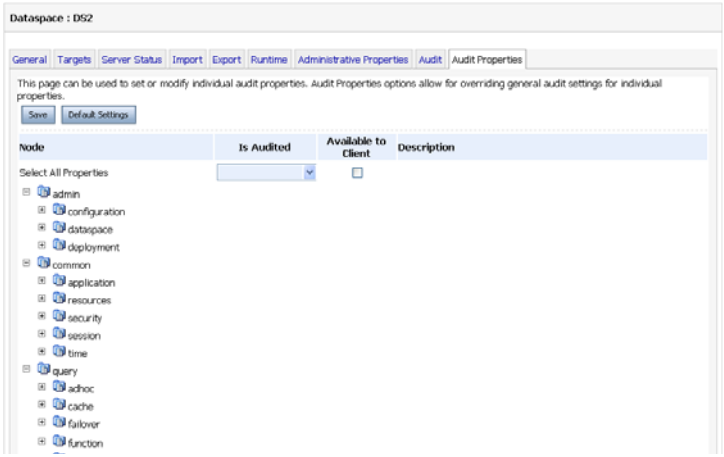
- Eclipse IDE is open with the ALDSP perspective and the dataspace has been successfully built and deployed.
- WebLogic Server is running.
- Your ALDSP Administration Console is open. In the sample dataspace the URI is:

  `http://localhost:7001/dspconsole`

- Auditing is enabled. (For details on activating and using auditing see Chapter 9, "Working With Audit and Log Information.")

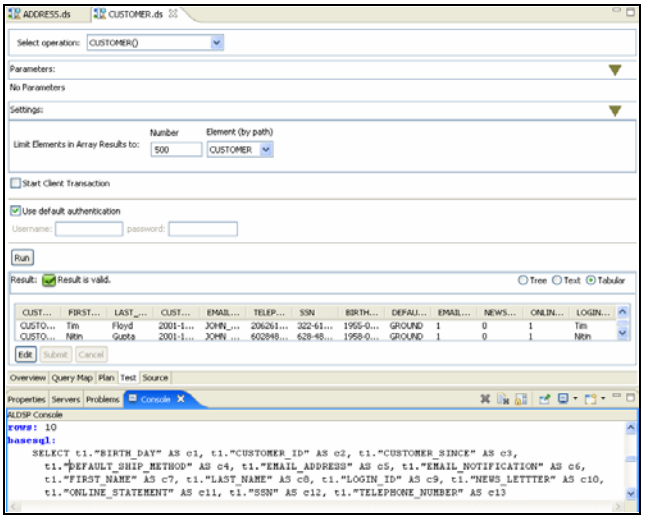
2. Set the *base SQL* statement audit property to Always (Figure 4-6), which means that the base SQL statement will always be returned. (See also "Setting Individual Auditing Properties" on page 9-5.)

Figure 4-6 Setting the basesql Property to Always be Returned



3. Select your relational data source in the ALDSP Administration Console (Figure 4-4).
4. Select the Substituted SQL statements option.
5. Click New and enter the following in the resulting dialog box:
   - Name you want to assign to your substitute query.
   - An optional description.
   - Enable (or disable) the substitution logic for the query you are about to create using the Enabled checkbox
6. Click Save > Activate Changes.
7. In your Eclipse IDE dataspace, run your query (such as CUSTOMER) in Test. Notice (Figure 4-7) that a basesql version of generated SQL statement is created.

Figure 4-7 Output from RTLApp CUSTOMER_ORDER() Query with basesql Result Highlighted



8. On the Console tab scroll down until you locate the basesql version of the query you just generated (also shown in Figure 4-7). Copy this version of the query to your clipboard. A sample query appears below:

```
SELECT t1."BIRTH_DAY" AS c1, t1."CUSTOMER_ID" AS c2, t1."CUSTOMER_SINCE"
AS c3,

      t1."DEFAULT_SHIP_METHOD" AS c4, t1."EMAIL_ADDRESS" AS c5,
t1."EMAIL_NOTIFICATION" AS c6,

     t1."FIRST_NAME" AS c7, t1."LAST_NAME" AS c8, t1."LOGIN_ID" AS c9,
t1."NEWS_LETTTER" AS c10,

    t1."ONLINE_STATEMENT" AS c11, t1."SSN" AS c12, t1."TELEPHONE_NUMBER"
AS c13

   FROM {RTLCUSTOMER}.{CUSTOMER} t1
```

9. Return to the ALDSP Administration Console, Substituted SQL Statements area and paste the basesql statement into the field labeled Generated SQL Statement.

10. Paste the basesql statement into the field labeled Substituted SQL statement.

11. Edit the substituted statement based on supported hints provided by the underlying database. A sample edited query restricting results to the first 10 rows in an Oracle database (emphasis added) — appears below:

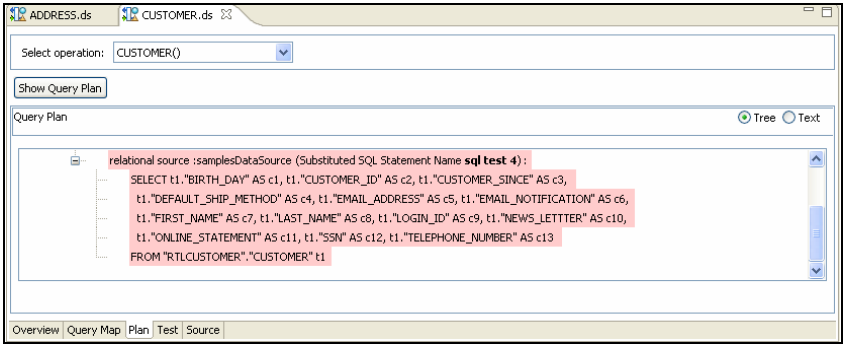
```
SELECT /*+ FIRST_ROWS (10)*/ t1."BIRTH_DAY" AS c1, t1."CUSTOMER_ID" AS
c2, t1."CUSTOMER_SINCE" AS c3,

      t1."DEFAULT_SHIP_METHOD" AS c4, t1."EMAIL_ADDRESS" AS c5,
t1."EMAIL_NOTIFICATION" AS c6,

      t1."FIRST_NAME" AS c7, t1."LAST_NAME" AS c8, t1."LOGIN_ID" AS c9,
t1."NEWS_LETTTER" AS c10,

    t1."ONLINE_STATEMENT" AS c11, t1."SSN" AS c12, t1."TELEPHONE_NUMBER"
AS c13

    FROM {RTLCUSTOMER}.{CUSTOMER} t1
```

12. Click Save > Activate Changes.

13. Return to the Eclipse IDE and re-run your query in Test mode. Notice in the Output pane that your substitute query appears in the SQL Statement area.

14. Select the CUSTOMER () query from the Plan view. Click Show Query Plan. Notice that the resulting plan contains the substituted SQL as well as the named of the substituted SQL statement.

Figure 4-8 Query Plan Displaying Substituted SQL Query



Setting the Server Resources

Configuring server resources optimally depends on the physical resources of the machine on which you deploy ALDSP, the anticipated load, and the type of dataspace you are deploying. Although the cached query plan count accelerates processing, it also consumes memory.

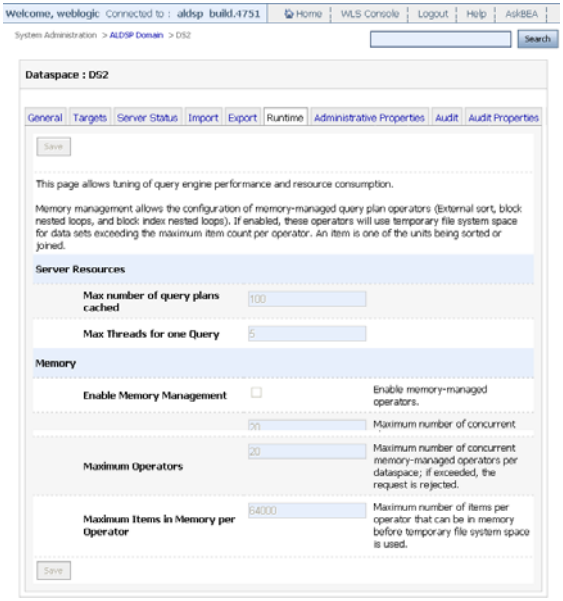
ALDSP Administration Console allows you to control server resources using the following options:

- Maximum number of query plans cached: This option sets the number of query plans that can be stored in cache for faster access.
- Maximum threads for one query: This option restricts the maximum number of parallel web service calls to the backend.

To set the server thread count:

1. Select the Runtime tab from the System Administration category.
2. Acquire the lock.
3. In the Server Resources section, specify the value for the maximum number of query plans cached and the maximum number of threads for a single query, as shown in Figure 4-9.

Figure 4-9 ALDSP Administration Console: Runtime Tab



4. Click Save > Activate Changes.

For more information on tuning performance for WebLogic Server, refer to the *WebLogic Server Performance and Tuning* guide.

Note: ALDSP 3.2 enables you to use Work Managers to manage the resources of a dataspace. See “Dataspace Deployment Using Work Managers” in the ALDSP 3.2 New Features Supplement

Item-based Memory Management

When memory management is enabled, ALDSP will use memory-managed sort and join operators. A memory-managed operator uses the disk to limit memory consumption in the presence of large datasets.

Each operator is only allowed to have up to a set maximum number of items in memory at a time. If the number of items to be processed exceeds the maximum then the operator must use the disk to complete its task. Here "items" are things that are being operated upon (joined or sorted).

Note: Different query workloads usually involve different size items.

For example, consider a query plan that contains 2 sort operators and 3 join operators. Assume that the maximum number of items per operator is 40,000. Regardless of the overall amount of data being processed by the query, this query plan will result in at most (2 + 3) * 40,000 = 200,000 items being held in memory at a time.

The maximum number of operators refers to the overall number of operators that may be concurrently running across all query plans being processed at a given time by the ALDSP-enabled server.

The maximum number of operators and the maximum number of items together provide a means to control the overall memory consumption of the server and can help guard against out-of-memory exceptions. When needed, these values should be adjusted based on workload and data characteristics, as the item count is only a coarse metric for memory consumption because item sizes affect the actual memory used as well.

To enable and configure memory management:

1. Click the Runtime tab from the System Administration category.
2. Acquire the lock.
3. From the Memory Management section (Figure 4-9), select Enable Memory Management.
4. Specify the limit for the maximum number of operators per dataspace using the Maximum Operators box. This allows you to restrict the memory usage by operators per dataspace.
5. Specify the limit for the maximum units that can be sorted or joined (items) by a single operator in memory. If this limit exceeds, then the item is stored in the temporary file system space.
6. Click Save > Activate Changes.

Using Administrative Properties

An administrative property is a user-defined property that you can configure using the ALDSP Administration Console. The value of an administrative property can be used in XQuery functions, either in data service functions or XQuery functions for security.

Note: For information on XQuery functions for security, see Chapter 5, "Securing ALDSP Resources."

An administrative property allows you to specify function parameters that can be easily changed by the administrator, without modifying the body of either the data service function or XQuery function for security.

Any data service within a dataspace can use the administrative property value. The property value can be accessed using XQuery with the BEA function `get-property()`. The function takes the name of the property as an argument and returns the value as a string. It also takes an argument that serves as the default value for the parameter. This value is used if the property is not configured in the console.

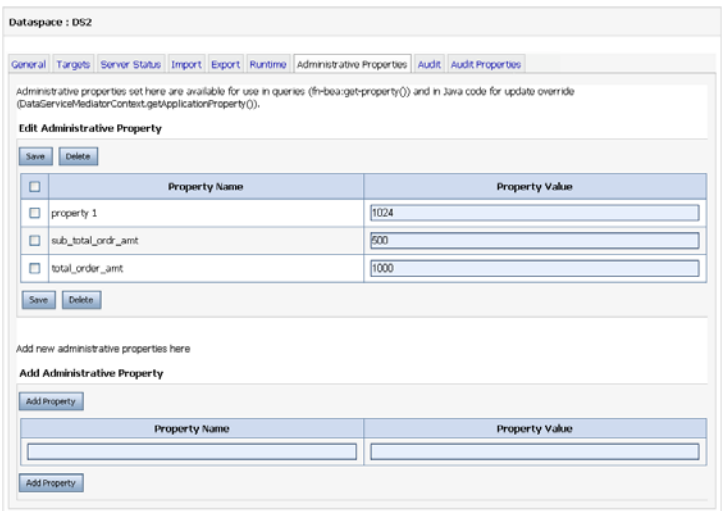
The following example illustrates an XQuery Function Library function that uses an administrative property:

```
declare function f1:getMaximumAccountViewable() as xsd:decimal {
   let $amount := fn-bea:get-property("maxAccountValue", "1000.00")
                                  cast as xsd:decimal
   return $amount
};
```

To manage administrative properties:

1. Click the name of the dataspace in the Navigation pane.
2. Click the Administrative Properties tab from the System Administration category. The list of property names currently defined appears in the table, as illustrated in Figure 4-10.

Figure 4-10 Administrative Properties Tab



3. Acquire the lock by selecting Lock & Edit.

4. To add a property, complete the following:

   a. Enter a name for the property in the Property Name field of the Add Administrative Property table.

      The name must match the name property passed to the `get-property()` function used to access the properties value. For example:

      ```
      fn-bea:get-property("maxAccountValue", "1")
      ```

   b. Optionally, enter an initial value for the property.

      You can change this value later, if required.

   c. Click Add Property.

      The property appears in the Edit Administrative Property table.

5. To change a property value:

   a. Acquire the lock.

   a. Enter a new value in the Property Value field of the Edit Administrative Property table.

   b. Click Save > Activate Changes.

6. To delete a property:

   a. Acquire the lock and select the property from the Edit Administrative Property table.

   a. Click Delete.

   b. Click Activate Changes to confirm deletion of the property.

Note: The default value for the property is used in any `get-property()` call using the deleted property.

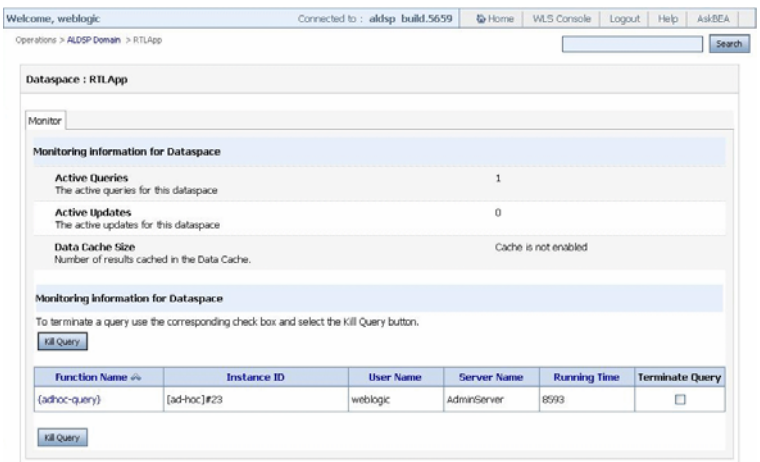
Monitoring Active Queries and Updates

Using the Operations category in the console, you can monitor long-running active queries and updates for a dataspace. The Operations category pertains to the runtime monitoring of deployed artifacts. In other words, the Operations category depends on the core (deployed) session. By contrast, other categories such as Service Explorer and Security relate to the session in progress.

Figure 4-11 illustrates an active ad hoc query running on the server for the RTLApp dataspace.

Note: Active queries and updates can be monitored only at the dataspace level.

Figure 4-11 Monitoring the Status of Active Ad Hoc Queries

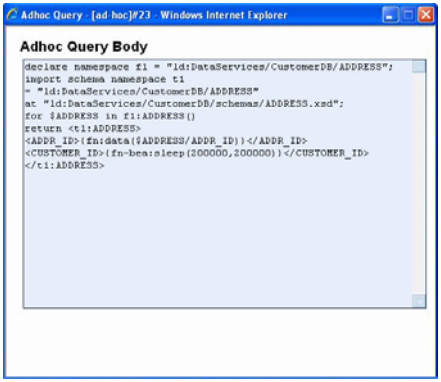


If an active query or an update is running for a long time on the server then the information is displayed in the table. This table lists the XQuery functions under the Function Name field.

If a query is taking longer than the expected time to retrieve data, you can also kill a query by clicking Kill Query.

In case of ad hoc queries, you can view the ad hoc query by clicking the function name in the Function field. This allows you to view the ad hoc query that is running on the server as shown in Figure 4-11.

Figure 4-12 Ad Hoc Query Displayed on ALDSP Administration Console



You can monitor active updates the same way as active queries.

Setting the Transaction Isolation Level

In some instances, ALDSP may not be able to read data from a database table because another dataspace has locked the table, causing queries issued by ALDSP to be queued until the dataspace releases the lock. To prevent this, you can set the transaction isolation to read uncommitted in the JDBC connection pool on your WebLogic Server.

To set the transaction isolation level:

1. Start the WebLogic Administration Console in a web browser by opening the following URL:

   `http://<HostName>:<Port>/console`

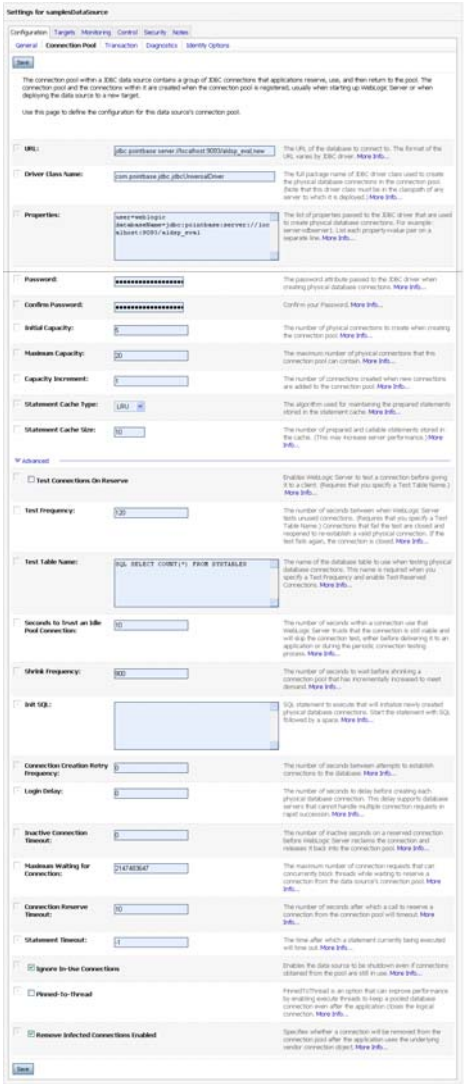
   For example, to start the Administration Console for a local instance of WebLogic Server (running on your own machine), type the following URL in a web browser address field:

   `http://localhost:7001/console/`

2. Expand Services > JDBC > Data Sources > <*datasourcename*>.

3. Select the Connection Pool tab as illustrated in Figure 4-13.

Figure 4-13 WebLogic Administration Console Connections Tab



4. Expand the Advanced section. The page expands to include the Advanced Options section.
5. Acquire the lock.
6. In the Init SQL field, enter the following:

   ```
   SQL SET TRANSACTION ISOLATION LEVEL READ UNCOMMITTED
   ```

7. Click Save > Activate Changes.

Preloading ALDSP Projects and Dataspaces

You can preload ALDSP projects and the dataspaces they contain whenever an ALDSP-enabled server is started by adding a property to the `setDomainEnv.cmd` file. If you have many projects and/or dataspaces, doing this can significantly improve initial Console performance.

To add this system property:

1. Stop the ALDSP-enable server if it is running.
2. Open the `setDomainEnv.cmd` file located in:
   `<aldsp_home>\samples\domains\aldsp\bin`

   where <*aldsp_home*> is the home directory for ALDSP, for example, `c:\bea\aldsp_3.0`
3. Add the following as a VM startup property:

   ```
   -Dcom.bea.dsp.oam.console.common.warmupTree=true
   ```

4. Save and close this file.
5. Start or restart your server.

CHAPTER 5

Securing ALDSP Resources

ALDSP 3.0 provides two types of security:

- **Managing Security at Runtime**: Runtime security enables you to define policies that secure ALDSP artifacts.
- **Controlling Administrative Access**: Access control policies enable restricting ALDSP Administration Console access based on user entitlements. Entitlements are predefined in the console and define the actions that a user can perform.

This chapter explains how you can configure and manage runtime security and access control for different users through the ALDSP Administration Console. It contains the following sections:

Introduction to ALDSP Security

To work with ALDSP security features, you must first define and create users who will access the ALDSP Administration Console. For more information about creating users, refer to Create Users in *WebLogic Server Administration Console Online Help*.

ALDSP 3.2 provides enhanced security support. See the ALDSP 3.2 New Features Supplement.

To secure ALDSP artifacts you can create runtime security policies. ALDSP artifacts or resources include dataspaces, services, operations, library procedures, and data elements.

For more information on runtime security policies, refer to Understanding Runtime Security Policies

After creating users in an ALDSP-enabled WebLogic Server domain, you can control administrative access of these users by applying administrative access control policies. Access control on ALDSP Administration Console is based on user entitlements. Entitlements are assigned to users by a domain user, who is a super user for a particular domain. A domain user is created when you create an ALDSP domain and specify the user name and password for it.

For more information on administrative access control, refer to Working with Administrative Access Control Policies.

Understanding Runtime Security Policies

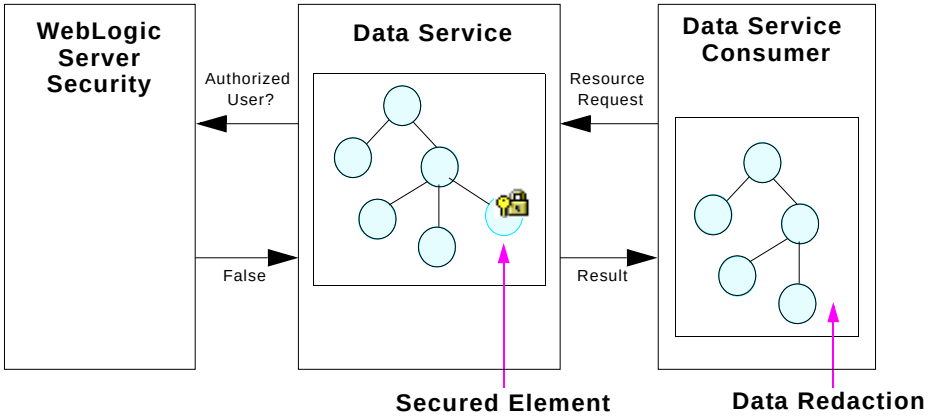
The runtime security feature enables you to configure access to resources such as dataspaces, data services, operations, and data elements. For a secured resource, a requesting client must meet the condition of the runtime security policy applicable to that resource, whether accessing the resource through the typed mediator API, an ad hoc query, or any data access interface. ALDSP exposes its deployed artifacts as resources that can be secured through runtime security policies.

For example, you can control access to an entire ALDSP dataspace or just to a credit card number element within Customer_Order.ds.

When a request comes to a running ALDSP instance for a secured resource, ALDSP passes an identifier for the resource to WebLogic Server. WebLogic Server, in turn, passes the resource identifier, user name, and other context information to the authorization provider, such as XACMLAuthorizer. The provider evaluates the policy that applies to the resource. As a result of the evaluation, access to the resource is either permitted or blocked.

If the user does not satisfy the requirements of an element-level policy, the element is *redacted* from the result object, and therefore does not appear.

Figure 5-1 Data Redaction



Note: By default, WebLogic Server security uses the XACML Authorization Provider. Other authenticators can use any external resource necessary to implement the policy evaluation.

Definition of a Securable Resource

A securable resource is an ALDSP artifact, such as a data service, operation, or element, to which you can apply a runtime security policy. The resources you can protect using runtime security include:

- **Dataspace:** The policies apply to all the resources in the dataspace. However, if there are policies applied to a data service or operation, then the more specific policy applies.
- **Data Service:** The policies apply to a data service and operations within that data service. However, if an individual operation has a policy applied to it, then the more specific policy applies.
- **Operations:** The policy applies to individual data service operations in a dataspace. Data service operations include ALDSP functions and procedures.
- **Data Elements:** A policy can apply to individual elements within a data service Return type, such as the salary property of a customer.

After you secure individual resources, you can enable or disable security for the dataspace. Security policies are inherited. This means that security enabled at the dataspace level applies to all data services, operations, and elements within the dataspace. However, if several policies apply to a particular resource, the more specific policy prevails. For example, a policy on an element supercedes a policy for the data service.

The hierarchy of ALDSP artifacts is as follows:

Dataspace

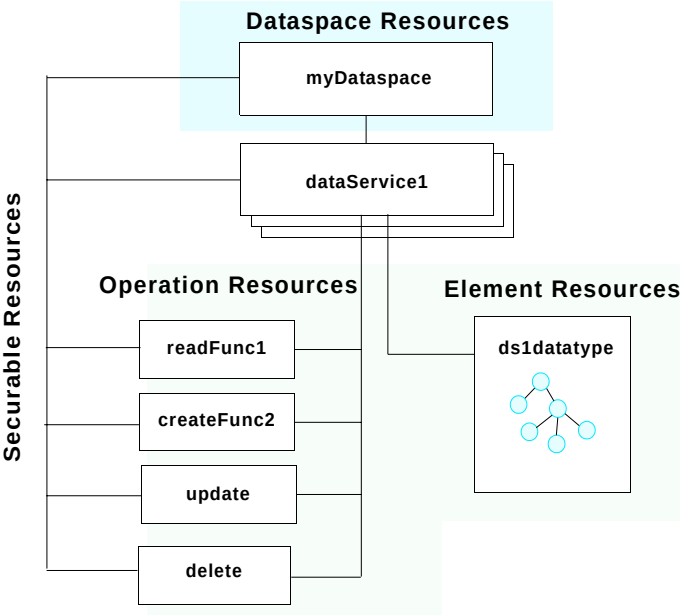
Data Service

Operations

Element

Figure 5-2 illustrates the securable resources in an ALDSP dataspace.

Figure 5-2 Securable Resources



Allowing Anonymous Access

At the dataspace level, you can enable anonymous access by creating a policy. If you apply this policy, all users, including unauthenticated users, can access resources by default. For more information on creating runtime policies at the dataspace level, refer to Configuring Dataspace-Level Security.

The anonymous access policy works only with the WebLogic Authorization provider. The ALDSP security policies are intended to work with the default WebLogic Authorization provider. If you are using another authorization provider, you will need to create policies using the facilities of the other provider. For more information, see WebLogic Authorization Provider: Provider Specific in the Administration Console Online Help.

The Security Configurations tab on ALDSP Administration Console provides the configurable runtime security policies. Setting up runtime security in ALDSP Administration Console involves the following tasks:

- Enabling Access Control
- Configuring security policies for dataspaces, data services, operations, and elements.
- Identifying data elements that you want to secure and then configure either security policies or custom XQuery security functions for the elements.

ALDSP directly supports runtime security policies for its resources. The WebLogic Platform supports extensive security features that can be applied to your implementation as well, including encryption-based, transport-level security. For runtime security configuration, ALDSP provides the following policies, called *predicates*, in ALDSP Administration Console:

- Role
- Group
- User
- Access occurs on specified days of the week
- Access occurs between specified hours
- Context element’s value is greater than a numeric constant

- Deny access to everyone
- Context element's value is equals a numeric constant
- Access occurs before
- Access occurs on the specified day of the month
- Context element's value equals a string constant
- Context element defined
- Allow access to everyone
- Access occurs after
- Access occurs before the specified day of the month
- Context element's value is less than the numeric constant
- Access occurs after the specified day of the month
- Server is in development mode

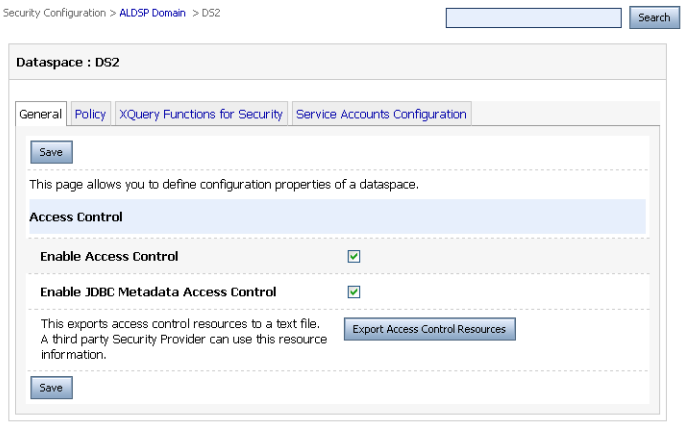
The security policies in the ALDSP Administration Console are similar to the conditions used by WebLogic Server security. For more information on WebLogic Server security policies and conditions, refer to "Securing WebLogic Resources Using Roles and Policies" in the WebLogic Server documentation.

In addition to creating runtime security policies, you can create service accounts to map security configurations of external data sources such as web services and Java functions. This feature ensures secure storage of the credentials of external data sources and allows runtime identity mapping.

Creating and Applying Runtime Security Policies

Before you start creating and applying runtime policies, make sure that the Enable Access Control checkbox in the General tab is selected, as shown in Figure 5-3. This activates the security policy configurations. If access control is not selected, then security is not enabled for your dataspace. The General tab is available only at the dataspace level.

Figure 5-3 General Tab



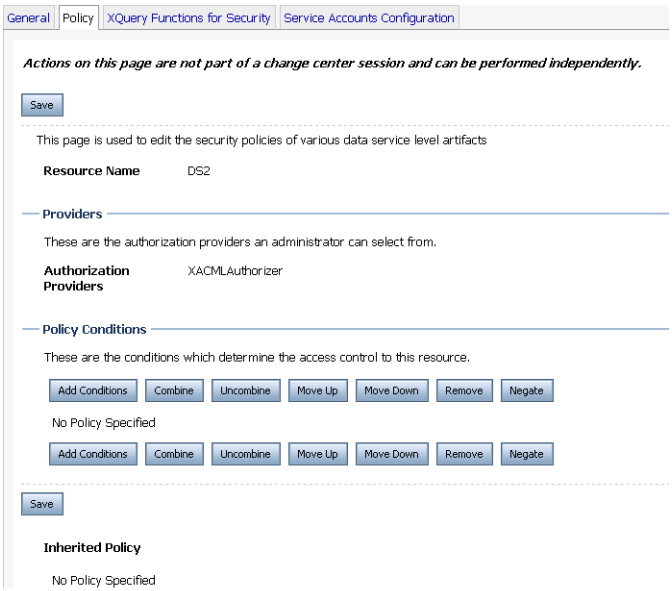
To enable access control:

1. Select the Security Configurations tab and the dataspace from the navigation pane.
2. Acquire the lock by clicking Lock & Edit.
3. Click the General tab.
4. Select Enable Access Control checkbox.
5. To enable JDBC metadata access, select Enable JDBC Metadata Access Control.
6. Click Save > Activate Changes.

The steps to create and apply runtime security policy for a dataspace, data service, and operations are the same. However, you must make sure that you select the ALDSP resource from the navigation pane. To create and apply the runtime security policy:

1. Select the Security Configuration category.
2. Click the Policy tab to start creating runtime policies for a dataspace, as shown in Figure 5-4.

Figure 5-4 Security Configurations: Policy Tab

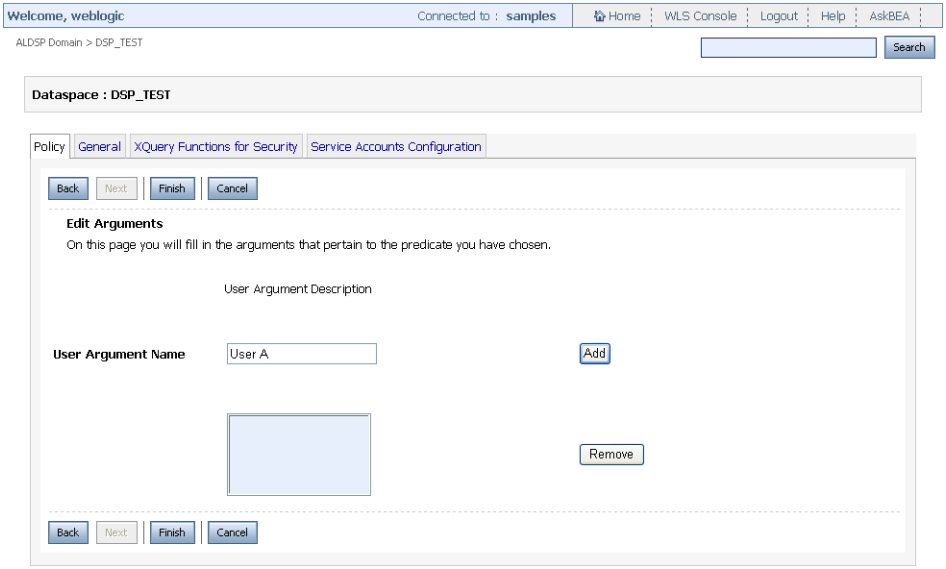


3. Click Add Conditions on the Policy tab. The Choose a Predicate page is displayed.
4. Select the predicate from the Predicate List drop down. For example, select User and click Next.
5. The next page that appears, depends on the predicate you select. If you select User predicate, the page show in Figure 5-5 is displayed.

Note: If you select the User predicate, it implies that you are allowing a particular user to access the dataspace. Make sure that this user is authenticated by WebLogic Server.

6. Specify the user name in the User Argument Name field, for example User A, and click Add. This adds the argument to the text box adjacent to the Remove button.

Figure 5-5 User Predicate Arguments Page



7. Click Finish. This adds the policy to the policy conditions applied to the dataspace.

Configuring Dataspace-Level Security

You can configure runtime policies that would ensure access to users who are assigned entitlements to access the entire dataspace. At the dataspace level, the Security Configuration tab provides the following tabs:

1. **General:** This tab provides the options to enable secured access to ALDSP resources and also to JDBC metadata. To access these options, click Lock & Edit to acquire the lock. It includes the following options:

- **Enabling Access Control:** Enabling access control activates checking security policies throughout the dataspaces within the domain. It ensures that access to any resource is determined by the policy on that resource.By default, access control is not enabled.
- **Enabling JDBC Metadata Access Control:** You can control metadata accessed through SQL by selecting the Enable JDBC Metadata Access Control option. This option allows ALDSP metadata access to users based on their access rights at the JDBC driver level. Selecting this option ensures that users are able to list only those tables and procedures that they are authorized to use. By default, this option is not enabled.

Note: If an access policy is time-dependent or is changed and the metadata access control option is enabled, you may not be able to access the tables and procedures that had been listed.

- **Export Access Control Resources:** This feature allows you to export the securable resource IDs within a dataspace to a text file format. However, it does not export the console configurations while exporting the ALDSP resources. This is helpful in determining the dataspace structure and defining policies on different systems, which may not be using the same authorization provider or are working on different servers.

  For more information, refer to Exporting Access Control Resources.

2. **Policy:** This tab allows you to edit policies if the default authorization provider, XACMLAuthorizer, is used. It provides the following information:

- **Resource Name:** The resource for which you need to add a runtime security policy.
- **Providers:** The authorization provider that WebLogic Server uses.
- **Policy Conditions:** List of policies that have been applied to the resource.
- **Overwritten Policy:** Any policy

  If a third-party authorization provider is used, then this tab displays a message as follows:

  "*Policies for AquaLogic Data Service Platform domain have to be defined in the configured external policy provider.*"

  For more information about creating and applying security policies, refer to Creating and Applying Runtime Security Policies.

3. **XQuery Functions for Security:** An XQuery function for security enables you to specify custom security policies that can be applied to data elements. In particular, security XQuery functions are useful for creating data-driven policies (policies based on data values). For example, you can block access to an element if the order amount exceeds a given threshold. For more information, refer to Working with XQuery Functions for Security.

4. **Service Accounts Configuration:** Service accounts represent a mapping of user credentials between an ALDSP user and the user of an external data source, such as a web service or Java function. This mapping is stored as a part of the dataspace configuration and ensures secure storage of external identity credentials. You can associate service accounts with a number of external data sources to perform runtime identity mapping. For more information, refer to Understanding and Using Service Accounts.

Working with XQuery Functions for Security

XQuery security functions allow data-driven security of ALDSP resources. At the dataspace level, you can create and maintain XQuery functions to ensure that data elements are returned only when the conditions are met. However, to associate these functions to data service elements, go to the data service and specify the element for which the function applies.

Note: If both a standard security policy and an XQuery function applies to a given data element, the results of both policy evaluations must be true for access to be permitted (logical *and* is applied to the results).

Applying data-driven security policies involves the following steps:

1. Identify the element as a secured element. (For more information, see “Configuring Data Element-level Security” on page 5-26.)
2. Create a security XQuery function to define the data-level security. (For more information, see “Creating an XQuery Function for Security” on page 5-11.)
3. Apply a security XQuery function to the data element. (For more information, see “Applying an XQuery Function for Security” on page 5-13.)

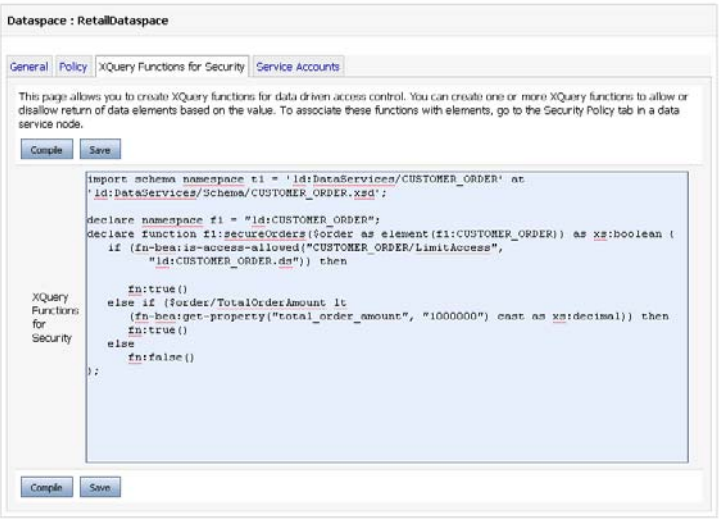
Creating an XQuery Function for Security

You can create one or more XQuery functions to apply to data elements within a dataspace.

To create an XQuery function for security:

1. Click Security Configurations tab and select the dataspace in the Navigation tree.
2. Click Lock & Edit to acquire the lock and then select the XQuery Functions for Security tab.

Figure 5-6 Security XQuery Functions



3. Add the XQuery function body in the text area of the tab, as shown in Figure 5-6. The following code sample is used in this illustration:

```
import schema namespace t1 = 'ld:DataServices/CUSTOMER_ORDER' at
   'ld:DataServices/Schema/CUSTOMER_ORDER.xsd';
declare namespace f1 = "ld:CUSTOMER_ORDER";
declare function f1:secureOrders($order as
       element(f1:CUSTOMER_ORDER)) as xs:boolean {
if (fn-bea:is-access-allowed("CUSTOMER_ORDER/LimitAccess",
       "ld:CUSTOMER_ORDER.ds")) then
   fn:true()
else if ($order/TotalOrderAmount lt
   (fn-bea:get-property("total_order_amount", "1000000") cast as
          xs:decimal)) then
       fn:true()
   else
       fn:false()
};
```

Notice that the function uses the BEA extension XQuery function `is-access-allowed()`. This function tests whether a user associated with the current request context can access the specified resource, which is denoted by an element name and a resource identifier.

ALDSP provides the following additional convenience functions for security purposes:

- `is-user-in-group ($arg as xs:string) as xs:boolean`

  Checks whether the current user is in the specified group.

- `is-user-in-role ($arg as xs:string) as xs:boolean`

  Convenience method that checks whether the current user is in the specified role.

- `userid() as xs:string`

  Returns the identifier of the user making the request for the protected resource.

Note: For details on creating XQuery functions, see the *XQuery and XQSE Developer's Guide*.

4. Click Compile and ensure that the function compiles successfully.
5. Click Save > Activate Changes to store the XQuery function.

Note: A security XQuery function must be applied to a data element for it to take effect. For more information, see "Applying an XQuery Function for Security" on page 5-13. The functions are applied to elements by qualified function name. The only requirement for the function is that it returns a Boolean value and that the name should be qualified by a namespace URI.

Applying an XQuery Function for Security

You can use XQuery functions for security to control access to data elements. After you define the XQuery function for security, as described in "Creating an XQuery Function for Security" on page 5-11, you must apply the function to the corresponding data element for it to take effect. In addition, you define policies for securing the data elements, which provide additional security along with the XQuery functions for security. For more information, refer to Configuring Data Element-level Security.

Note: ALDSP 3.2 provides support for data redaction behavior for data elements. See "Encryption-based Data Redaction" in the ALDSP 3.2 New Features Supplement.

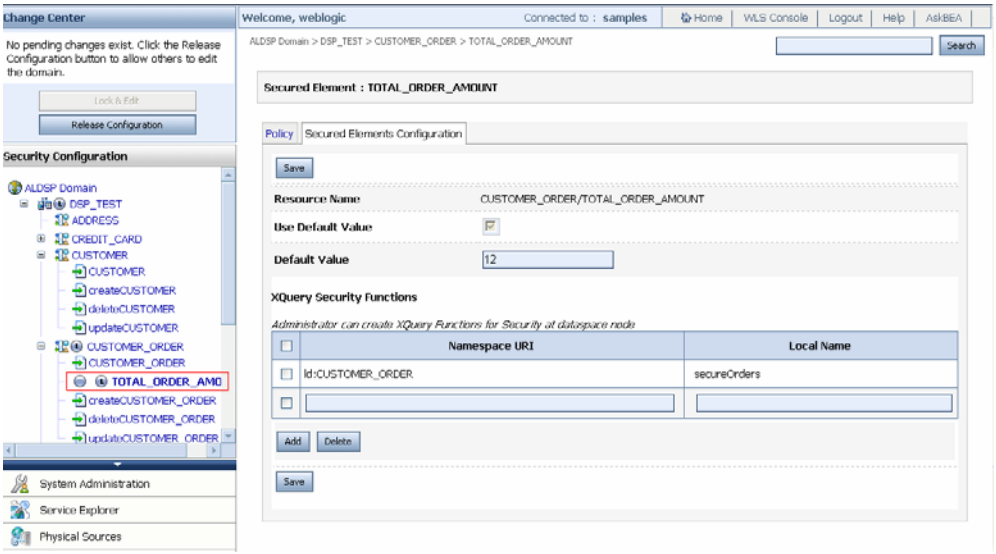
To make any changes to the security configurations of a data element, you must first acquire the lock by clicking Lock & Edit. To apply the XQuery function for security to a data element:

1. Select the Security Configuration tab from the navigation pane and then click the data service associated with the data element that you need to secure.
2. Click the Secured Elements tab and select the checkbox next to the data element to which you want to apply a custom function.

3. Click Save and then click Activate Changes. This data element is now visible under the data service in the navigation tree.

4. Select the data element from the navigation tree and click the Secured Elements Configuration tab. This tab allows you to specify the qualified function name and namespace URI for the XQuery function that you want to associate with the data element, as shown in Figure 5-7.

Figure 5-7 Applying XQuery Functions for Security



5. If you want to specify a default value for the element or attribute, then select the User Default Value checkbox and specify the default value in the Default Value box.

   This option allows you to assign a constant value for the element or attribute. However, it supports only primitive types, so you cannot have a default value for complex types.

   Note: If you select this check box, then it is mandatory to specify the default value for the resource.

6. Specify the namespace URI and local name of the XQuery function that you have created.

7. Click Add > Save > Activate Changes. This completes the association of the data element with the XQuery function for security.

Understanding and Using Service Accounts

Service accounts provide the option to store user credentials for external data sources. It provides a mapping between the local WebLogic user and a remote external data source user by configuring the user credentials within the ALDSP Administration Console.

You can configure service accounts for web services and Java functions. For JDBC identity mapping, ALDSP depends on WebLogic Server 9.2 and 10.0 built-in support.

Service accounts provide different types of mappings, which include:

- **Static:** This mapping option enables you to map all ALDSP users, including unauthenticated users, to a single external data source user.
- **Mapping:** This option enables you to create a mapping of an ALDSP user to an external data source user. You can also map multiple ALDSP users to a single external data source user. For unauthenticated users you may define a mapping, otherwise an error will occur when the unauthenticated user tries to access ALDSP.
- **Identity Mapping:** This option enables you to create a mapping between external data source users and identically-named authenticated ALDSP users, supplying the passwords of only the external data source users.

Note: After you create and define the type of a service account, you cannot change it. If you have to change a service account type, delete the account and create a new one.

Creating a Service Account

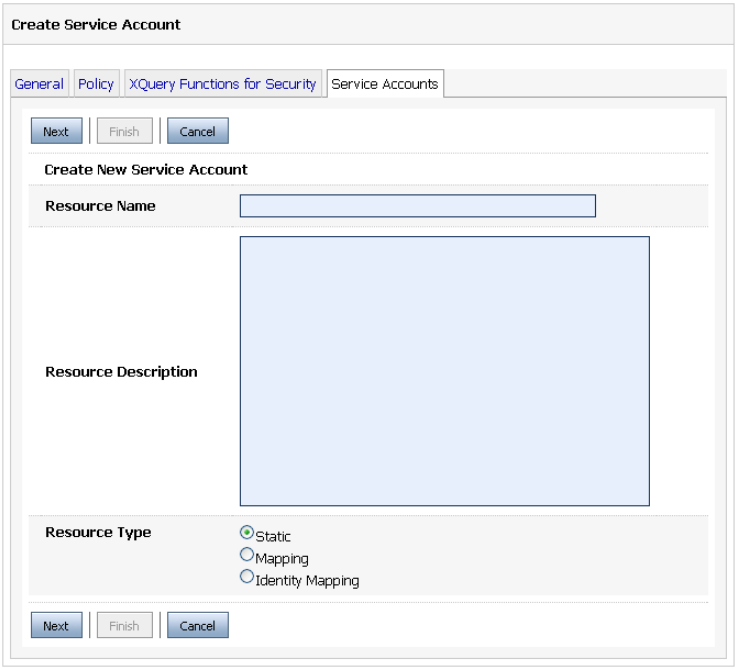
To create a service account:

1. Click the Security Configurations tab in the category list, select the dataspace in the navigation tree, and click the Service Accounts tab in the workspace content area.
2. Click Lock & Edit to acquire the lock.
3. Click New. This opens the Create a New Service Account page, as shown in Figure 5-8.

4. On this page, specify the following details:

- Resource Name: Name of the service account.
- Resource Description: A description of the service account. This is optional.
- Resource Type: Select the type of the service account from the list of available options including Static, Mapping, and Identity Mapping.

Figure 5-8 Create a New Service Account Page



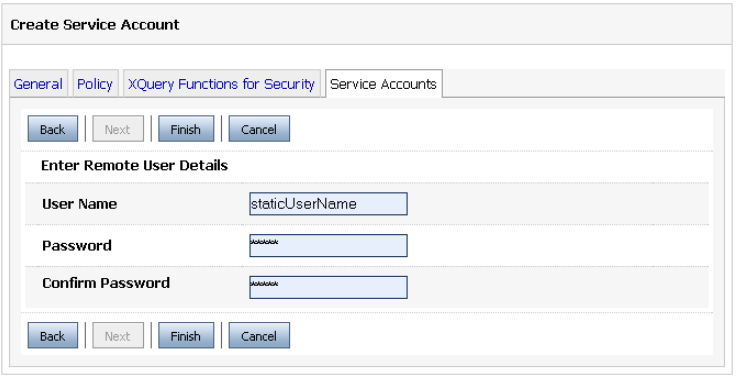
Note: Based on the selected resource type, the Next button is enabled.

5. If you select the resource type as Static:

   a. Click Next.

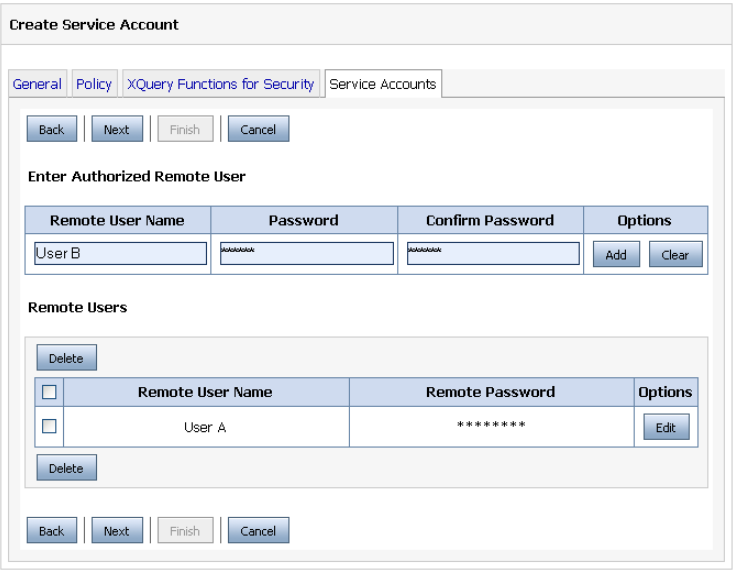
   b. On the next page, specify the user name and password for that account and click Finish, as shown in Figure 5-9.

Figure 5-9 Creating a Static Service Account



6. If you select the resource type as Mapping and click Next, a new page is displayed, as shown in Figure 5-10.

Figure 5-10 Creating a Service Account of the Mapping Type



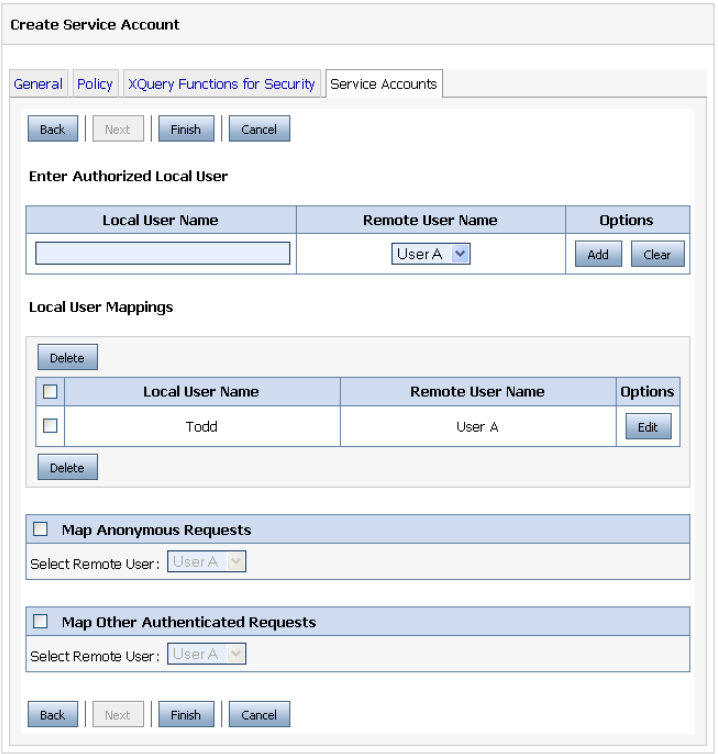
On this page, you can define the remote (external data source) users.

a. Specify the remote user name and password in the Remote User Name and Password fields, respectively, of the Enter Authorized Remote User table.

b. Click Add. This adds the users to the Remote Users table. Using the Remote Users table you can edit the password or delete a user, as required.

c. Click Next after adding the remote users. The next page enables you map the local users to remote users, as shown in Figure 5-11.

Figure 5-11 Local User to Remote Mapping



d. Specify the local user name in the Local User Name field and select the corresponding remote user from the Remote User Name list.

e. Click Add. This creates the local to remote user mapping.

f. To map all unauthenticated (anonymous) users to a particular remote user, click the Map Anonymous Requests checkbox and then choose the remote user from the drop-down list.

g. In case you want to provide a default mapping for all authenticated user that do not have an explicit mapping to the remote user, click the Map Other Authenticated Requests checkbox and choose the remote user from the drop-down list.

h. Click Finish.

7. If you select the resource type as Identity Mapping and click Next, a page is displayed, as shown in Figure 5-11. This page is identical to the page displayed when you select Mapping as the resource type.

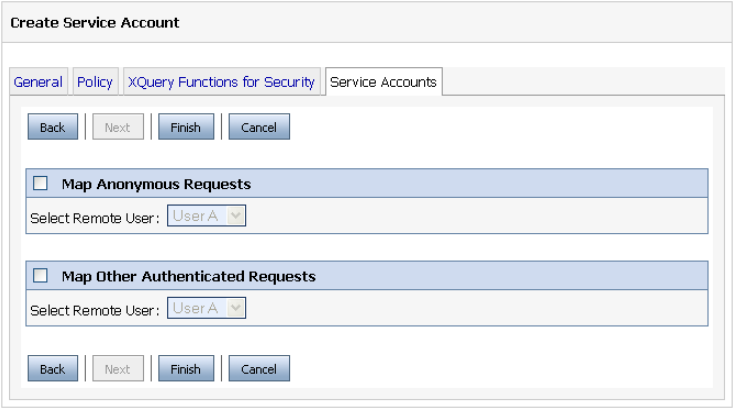
   On this page, you can define the authorized remote (external data source) users, and add them as authenticated external data source users.

   a. Specify the remote user name and password in the Remote User Name and Password fields, respectively, of the Enter Authorized Remote User table.

   b. Click Add. This adds the users to the Remote Users table. Using the Remote Users table you can edit the password or delete a user, as required.

   c. Click Next after adding the remote users. The next page enables you to map anonymous requests or other authenticated requests to remote users.

Figure 5-12 Mapping Anonymous Requests or Other Authenticated Requests



d. To map all unauthenticated (anonymous) users to a particular remote user, click the Map Anonymous Requests checkbox and then choose the remote user from the drop-down list.

e. In case you want to provide a default mapping for all authenticated user that do not have an explicit mapping to the remote user, click the Map Other Authenticated Requests checkbox and choose the remote user from the drop-down list.

f. Click Finish. This creates a mapping between the defined external data source users and the identically-named authenticated ALDSP users.

8. Click Activate Changes.

Exporting Access Control Resources

Authorization is the process whereby the interaction between users and resources are limited to ensure integrity, confidentiality, and availability. WebLogic uses resource identifiers to identify deployed ALDSP artifacts, such as dataspaces, data services, and operations. This identifier is used to associate a client request to any security policies configured for the requested resource.

Resource identifiers are managed for you when you use the default WebLogic Authorization provider and the ALDSP Administration Console to configure your policies. In particular, resource identifiers already exist for ALDSP dataspaces, data services, and data service operations. In addition, when you choose elements to be secured in the console, an identifier is generated for the element.

However, when using a custom authorizer, you must know the resource identifiers for your deployment and configure policies for the resources in the form expected by the other authorization module. This means that you need to identify the element resources that need to be protected.

Note: The WebLogic security documentation provides details on how to connect another security authenticator to WebLogic Server. For more information, see *WebLogic Authorization Provider* in the *Administration Console Online Help*.

You can view the list of resource identifiers by exporting the access control resources from the ALDSP Administration Console.

To export the file:

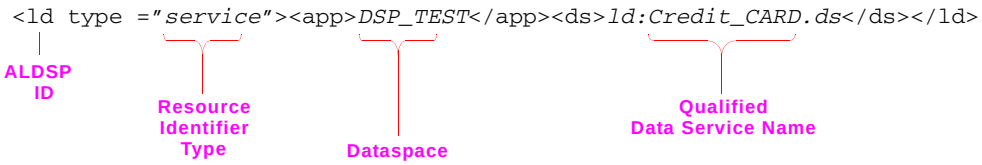
1. Select the dataspace in the navigation pane and select the General tab from the Security Configuration category.
2. Click Lock & Edit and then click Export Access Control Resources if you want to export the current session values of the dataspace.
3. If you want to export the core values, then click Export Access Control Resources without acquiring the lock.
4. Save the text file.

   An example of a portion of the file follows:

```
<ld type="admin"><app>DOMAIN</app></ld>

<ld type="admin"><app>ADMIN</app></ld>

<ld type="admin"><app>MONITOR</app></ld>

<ld type="admin"><app>BROWSER</app></ld>

<ld type="admin"><app>ADMIN</app><ds>DSP_TEST</ds></ld>

<ld type="admin"><app>MONITOR</app><ds>DSP_TEST</ds></ld>

<ld type="admin"><app>BROWSER</app><ds>DSP_TEST</ds></ld>

<ld type="app"><app>DSP_TEST</app></ld>

<ld type="service"><app>DSP_TEST</app><ds>ld:CREDIT_CARD.ds</ds></ld>

<ld
type="function"><app>DSP_TEST</app><ds>ld:CREDIT_CARD.ds</ds><res>{ld:C
REDIT_CARD}CREDIT_CARD:0</res></ld>

<ld
type="function"><app>DSP_TEST</app><ds>ld:CREDIT_CARD.ds</ds><res>{ld:C
REDIT_CARD}createCREDIT_CARD:1</res></ld>

<ld
type="function"><app>DSP_TEST</app><ds>ld:CREDIT_CARD.ds</ds><res>{ld:C
REDIT_CARD}deleteCREDIT_CARD:1</res></ld>

<ld
type="function"><app>DSP_TEST</app><ds>ld:CREDIT_CARD.ds</ds><res>{ld:C
REDIT_CARD}updateCREDIT_CARD:1</res></ld>

<ld type="service"><app>DSP_TEST</app><ds>ld:CUSTOMER.ds</ds></ld>
```

The format of a resource identifier is shown in Figure 5-13.

Figure 5-13 Resource Identifier Format



The type can be admin, service, or function. The resource can be any of the following:

- **Function:** A data service function, for example, `{ld:DataServices/ElectronicsWS/getProductList}getProductList:1`
- **User-defined or administrative entity:** A custom entity, such as a protected element or an arbitrary label defined in a data service that is used with fn-bea:is-access-allowed operation.

These are generated when you select an element in the Secured Element tab of the ALDSP Administration Console.

Configuring Data Service and Operation-Level Security

A data service has several operations, including one or more read, create, update, delete, navigation, and library operations. The security policies that you apply at the data service level apply to data service operations and data elements. You can also select the data elements that you want to secure at the data service level.

Operation-level security policies enable you to control:

- User access to data service operations. It enables you to set stricter controls on the ability to change data, for example, compared to the ability to read data.
- Access time of data service operations. Enables you to control the time when a particular operation can or cannot be accessed.

Note: Make sure that you configure policies on the data service resources that are accessed directly by the user. Security policies on data services that are used by other data services are not inherited by the calling data service. This means that if a data service with a secured resource is accessed through another data service, the policy is not evaluated against the caller. For more information, refer to Creating and Configuring Security Policies for Operations.

Data service operations are identified by name and number of parameters for setting security configurations. If you modify the number of parameters, you will need to reconfigure the security settings for the operation.

Creating Data Service Runtime Security Policies

The steps to create the security policy at the data service and operation level are the same as the dataspace level. Refer to Creating and Applying Runtime Security Policies for details.

Note: ALDSP 3.2 provides enhanced security support. See the ALDSP 3.2 New Features Supplement.

At the data service level, you can select all the data elements in a data service by selecting the top-level element (Customers in Figure 5-14) or individual data elements that you want to secure using the Secured Elements tab.

For example, if you create an XQuery function for security and you want to associate it with a data element, you can select the data element from the Secured Elements tab and then configure the data-element level security. (For more information about XQuery function for security, refer to Working with XQuery Functions for Security.)

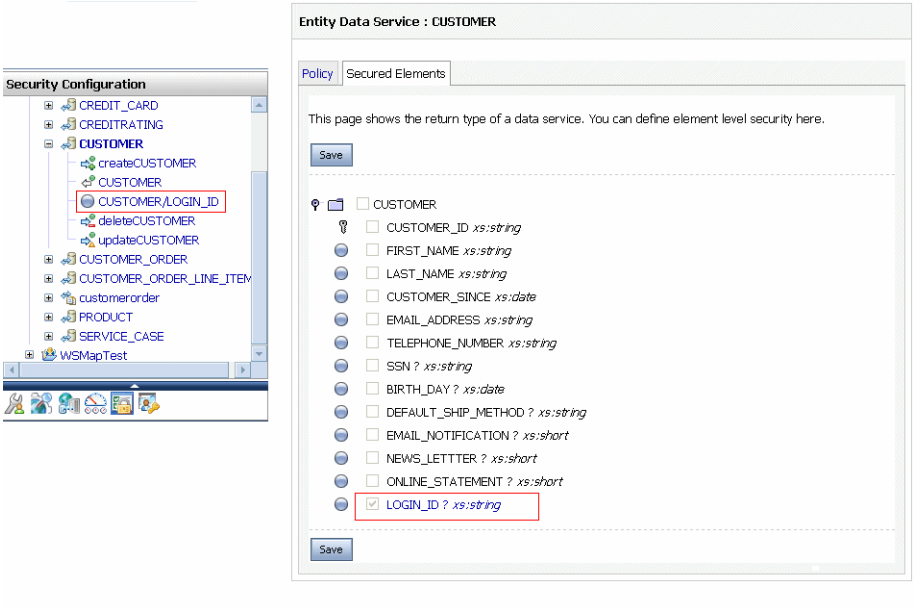
Note: You should only secure the root element of a data service if you are confident that none of the elements used by read functions in the service must return a value. Since a secured element does not return a value, a schema which requires that one or more values be returned will fail with a runtime error. Alternatively, you can modify the schema so that elements are optionally returned.

To select the data element to be secured:

1. Acquire the lock and select the data service.
2. Select the Secured Elements tab, as shown in Figure 5-14.
3. Select the data element that you want to configure for security.

4. Click Save > Activate Changes. Notice that the selected element is now included in the navigation tree under the data service, as shown in Figure 5-14.

Figure 5-14 Secured Data Element in the Navigation Tree



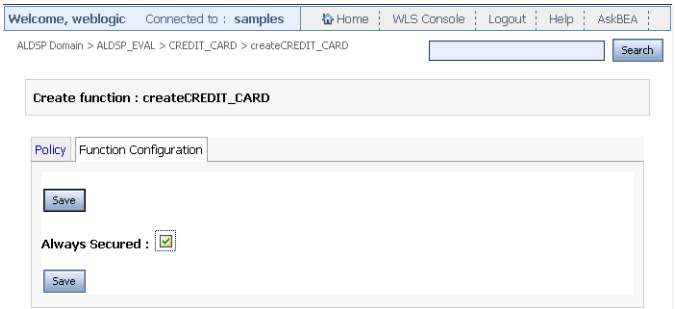
To apply security policy to the data element, select the element from the navigation tree. You can also select the secured element using the Secured Elements tab. For more information, refer to Configuring Data Element-level Security.

Creating and Configuring Security Policies for Operations

To set runtime security policy for an operation:

1. Select the operation from the navigation tree and click the Function Configuration tab.
2. Select the Always Secured checkbox and click Save as shown in Figure 5-15.

Figure 5-15 Function Configuration Tab



This setting ensures that every time this operation is accessed, the runtime policy is adhered to. Consider the following example:

- Operation 1 (fn1) has a runtime policy to allow access to user1 or user2.
- Operation 2 (fn2) has a runtime policy to allow access to user2 only and the operation configuration is set to Always Secured.
- fn1 invokes fn2.

In this scenario, if you access fn1 using user1, then access will be denied because the runtime security policy configuration does not allow user1 to access fn2.

If you do not select the Always Secured check box for fn2, then you will be able to access fn1 if using either user1 or user2 because the system will check the security policy for fn1 only and not fn2.

Configuring Data Element-level Security

Element-level security associates a security policy with a data element for the Return type within a data service. If the policy condition is not met, the corresponding data is not included in the result.

When configuring element-level security, you first identify the element as a securable resource, then set a policy on the resource.

The data element security policy can be configured using the steps described in Creating and Applying Runtime Security Policies.

To associate an XQuery function for security with a corresponding data element, select the Secured Elements Configuration tab and follow the steps mentioned in Applying an XQuery Function for Security.

Note: Element-level security is only applied when all of the following conditions are met:

- The data is being delivered across the “client-server boundary”.
- The security is applied to a data service that is directly accessed by a valid client process. In other words, element-level security policies are not “inherited” from underlying or invoked data services.
- The element being secured is accessed from a client using a read or navigation operation or an ad hoc query.

Additional Data Element Security Considerations

To ensure the security of elements, you need to manage and layer data services properly. This means being careful not to create insecure holes in the layers and not depending on security settings for data services which are not being directly invoked by the client.

Note: Secured elements, in general, should never offer public read or navigate functions that accept a secured element value as an input argument as this can permit guessing-style attacks to reveal otherwise secure data.

Securing Native Web Services

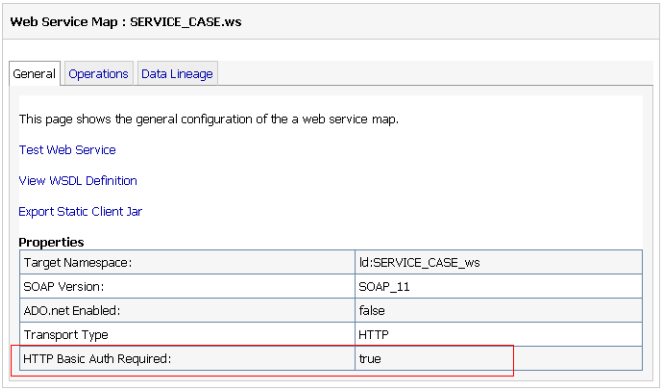
You can set the security policies for native web services using the Basic Auth Required property in the Eclipse IDE. You can create runtime security policies for a native web service and then set this property to true. This applies the security policy for the native web service. For more information about the Basic Auth Required property, refer to the Add Security Resources to Data Services topic in the Designing Logical Data Services section of the *Data Services Developer's Guide*.

The Service Explorer in ALDSP Administration Console allows you to check if the Basic Auth Required property is set to true or false.

To view information about this property in the Service Explorer:

1. Click the Service Explorer category. The General tab is displayed as shown in Figure 5-16.

Figure 5-16 Basic Auth Required Property Information in Service Explorer

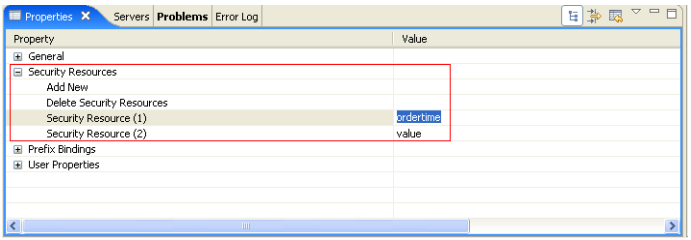


2. Select the native web service from the navigation tree. In this case, the Basic Auth Required property is set to true. This implies that some security policy is applied to SERVICE_CASE.ws, which the native web service.

Creating Security Policies for User-Defined Security Resources

User-defined security resources are created in the Eclipse IDE Property Editor, as shown in Figure 5-17.

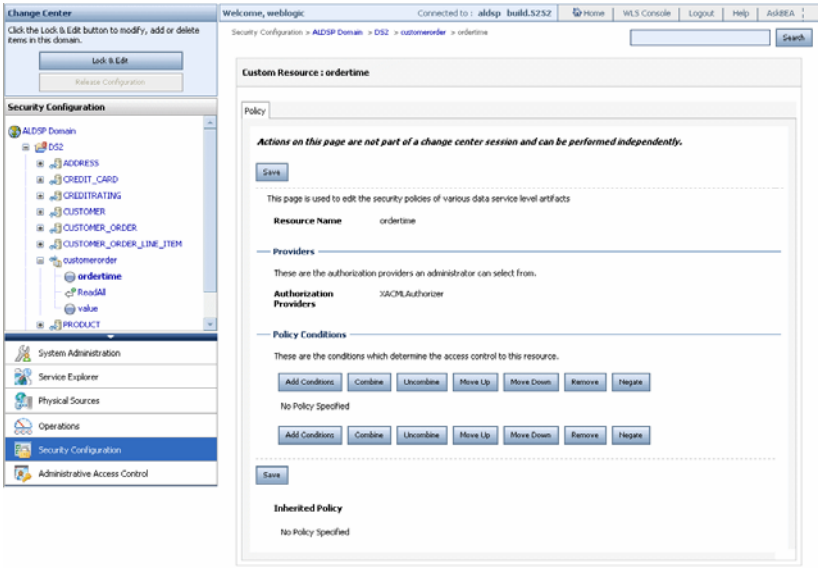
Figure 5-17 ALDSP IDE Property Editor: User-Defined Security Resources



For more information about setting the security resource values, refer to Declare a Security Resource in *Data Services Developer's Guide*.

After you assign a value to the security resource, you can create runtime security policies for the user-defined security resource. In the preceding figure, ordertime is the value of the security resource. After you deploy the dataspace, this resource is displayed in ALDSP Administration Console. Figure 5-18 shows the ordertime security resource in the navigation tree for the customerorder data service.

Figure 5-18 Creating Runtime Security Policy for a User-Defined Security Resource

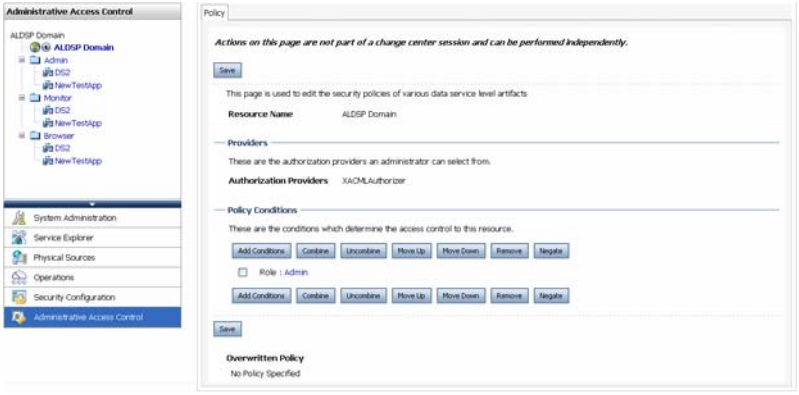


You can create a runtime security policy for the ordertime security resource using the console.

Working with Administrative Access Control Policies

Administrative roles require entitlements to access ALDSP Administration Console. These entitlements can be assigned through the Administrative Access Control category, as shown in Figure 5-19.

Figure 5-19 Administrative Access Control Tab



A domain user, who is the super user for the console, assigns entitlements to users. In addition to the domain entitlement, other predefined entitlements are admin, monitor, and browser, which allow access to information for different categories and resources. The hierarchical structure of the entitlements is as follows:

Domain

Admin

Monitor

Browser

This hierarchy implies that the domain entitlement allows you to perform all the tasks on ALDSP Administration Console, depending on whether the domain entitlement is for all the dataspaces within a domain or a particular dataspace. However, other entitlements cannot perform all the tasks that can be performed by a user with domain entitlement.

For example, you can set the administrative access control policies only if you have domain entitlement. Similarly, the admin entitlement allows you to perform more tasks on a dataspace than monitor or browser entitlements.

Note: Entitlements can be assigned at the dataspace level or for all the dataspaces. For example, for User A, you can assign admin entitlement for DS1, monitor entitlement for DS2, and browser entitlement for DS3. Alternatively, you can assign the Admin entitlement for all the dataspaces within the domain to User A. For more information, refer to Assigning Entitlements.

A default domain user is created on WebLogic Server when you create the ALDSP domain. There can be more than one domain user for the console and one domain user can create other domain users.

Note:

By default, an Admin role is created for a domain user in ALDSP Administration Console which is mapped from WebLogic Server Administrator role, as shown in Figure 5-19.

Table 5-1 lists the tasks that can be performed by a user for each entitlement.

Table 5-1 Tasks Allowed for Entitlements

Entitlement	Categories and Resources Available
Domain	The domain user for a dataspace can perform all the tasks on the ALDSP Administration Console. Some of the most important tasks that a domain user can perform include the following: • Creating, deploying, and deleting dataspaces • Creating users with domain, admin, monitor, browser entitlements • Editing and updating configurations • Acquiring lock from a user forcibly • Viewing all tabs in the category list, including the Administrative Access Control tab • Configuring auditing options • Manage data cache **Note:** Only a domain user can acquire a lock forcibly from another user, regardless of the user entitlement. This means that the one domain user can forcibly acquire the lock from another domain user also.

Table 5-1 Tasks Allowed for Entitlements (Continued)

Entitlement	Categories and Resources Available
Admin	Most of the information available to an admin user for a dataspace is the same as the domain user. However, an admin user cannot create or delete dataspaces and cannot assign entitlements. Therefore, when you log into the console with admin entitlement, then the Administrative Access Control tab will not be available.
Monitor	A monitor for a dataspace cannot make any changes in the ALDSP Administration Console. Therefore, the change center is disabled for the dataspace for which the user has monitor entitlements. The System Administration tab for a monitor user does not provide any options. A monitor user can view the following on the console: • Data cache, queries and updates available through the Operations category • For the dataspace, a monitor user can export the static mediator client jar file using the General tab.
Browser	A browser user has the least control over the ALDSP Administration Console. This user entitlement can only browse through the console. The change center is disabled for this user. However, like a monitor user, a browser user can also export the static mediator client JAR file.

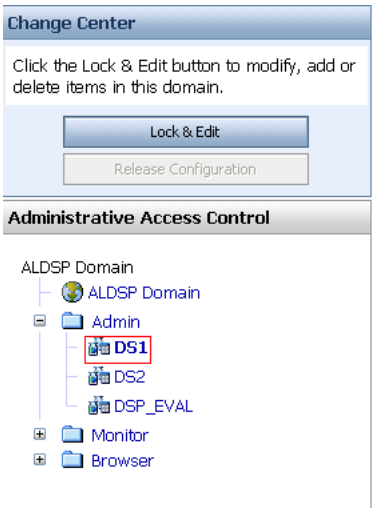
Assigning Entitlements

Entitlements are created for users that are created on WebLogic Server 9.2 and can be managed through the WebLogic Server Administration Console.

To assign an entitlement:

1. Log into the ALDSP Administration Console using the domain user name and password.
2. Select the Administrative Access Control category.
3. If you want to assign entitlement for a specific dataspace, then from the navigation tree, select the dataspace listed under the entitlement. For example, if you want to assign admin entitlement for dataspace DS1, then select DS1 listed under the Admin entitlement, as shown in Figure 5-20.

Figure 5-20 Assigning Entitlement for a Dataspace



You can also assign an entitlement to a user for all dataspaces within the domain. For example, if you want to assign the Admin entitlement for dataspaces DS1, DS2, and DS3 to a user, then select the Admin entitlement option. Similarly, you can assign, monitor and browser entitlements to a user for all dataspaces by selecting the Monitor or Browser option from the navigation tree.

Note: In this case, the Admin entitlement is selected for the dataspace DS1.

4. Click Add Conditions on the Policy tab.

5. Select the predicate as User and click Next.

Note: You can also select other options from the list of predicates. For more information, refer to Understanding Runtime Security Policies.

6. Specify the user name for which you want to assign the admin entitlement and click Finish. This creates a user who has Admin entitlement for dataspace DS1.

A user views the category-list based on the entitlement assigned to that user for that dataspace. For example, User A with admin entitlement for DS1 can view the Security Configurations tab, however, if User A has monitor entitlement for DS2, then the Security Configuration tab for DS2 will not appear for User A.

Gaining Administrative Access After a System Lockout

Security policies configured for assigning Admin entitlement to a user may get deleted inadvertently. If that is the only Admin user entitlement for ALDSP Administration Console, then the Admin user is locked out of the console.

In this case, you can configure the `com.bea.dsp.security.admin.bootstrap` system property for WebLogic Server. This property allows you to specify a user name, who gains domain access rights. However, this property should only be used if the ALDSP Administration Console is locked due to some policy editing.

To configure this system property:

1. Stop WebLogic Server.
2. Open the `setDomainEnv.cmd` file located in:
   `<aldsp_home>\samples\domains\aldsp\bin`

   where <aldsp_home> is the home directory for ALDSP, for example, `c:\bea\aldsp_3.0`
3. Edit this file to include the `com.bea.dsp.security.admin.bootstrap` system property. For example:

   ```
   set JAVA_OPTIONS=%JAVA_OPTIONS% %JAVA_PROPERTIES%
   -Dwlw.iterativeDev=%iterativeDevFlag%
   -Dwlw.testConsole=%testConsoleFlag%
   -Dwlw.logErrorsToConsole=%logErrorsToConsoleFlag%
   -Dcom.bea.dsp.security.admin.bootstrap=<username>
   ```

   where <username> is the place to specify the Admin user for ALDSP Administration Console.

Note: The user name specified in the `com.bea.dsp.security.admin.bootstrap` system property should be a user that has already been created using the WebLogic Server console.

4. Save and close this file.
5. Restart WebLogic Server.
6. Log in to ALDSP Administration Console using this user name and then re-configure the Admin entitlement policies.

Taking Lock and Edit Capability

A domain user can take back the control of the lock from ALDSP Administration Console. The lock may need to be taken back from a user in cases where a user, such as an admin user, has acquired the lock but has not released it for a long period and another admin user needs to acquire the lock to modify configurations. One domain user can acquire the lock from another domain user also.

When lock is acquired by a user, the Take Lock & Edit option is enabled for the domain user as shown in Figure 5-21.

Figure 5-21 Take Lock & Edit Enabled in the Change Center

The domain user can click the Take Lock & Edit option from the change center to acquire the lock. In this case, the user whose lock is acquired will see the core configuration values on the console and the domain user or the other admin user will be able to view all the changes made by the other user using the pending changelist. For more information about pending changelist, refer to "Pending Changelist" on page 2-9.

CHAPTER 6

Viewing Native Web Services

A native web service is a data service that is exposed as a web service through ALDSP. It allows a direct mapping from the data service to the web service and updates configurations at runtime. To generate a native web service the system requires a web service map file, which is used to generate the WSDL for the web service. A web service map file describes the mapping between the data services, functions, and WSDL operations.

- For more information about creating a native web service, refer to How To Generate a Web Service Map from a Data Service in the *Data Services Developer's Guide*.
- For information about consuming a native web service, refer to Invoking Data Services Through Web Services in the *Client Application Developer's Guide*.

ALDSP Administration Console displays the web service map artifacts in the dataspace through the Service Explorer.

This chapter describes the steps to view the artifacts for the web service and the WSDL definition, and export it using ALDSP Administration Console.

Viewing Native Web Service Artifacts

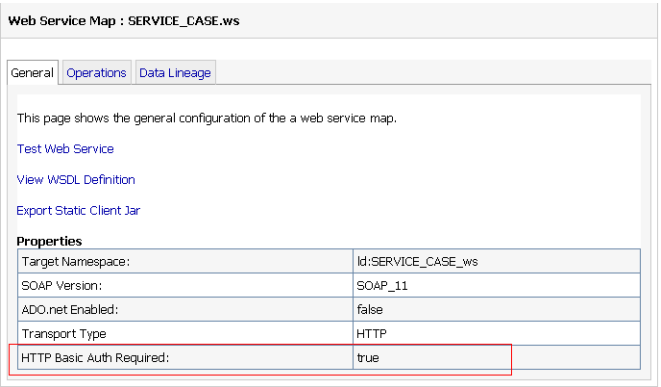
When you click the Service Explorer category for a web service, the following tabs are displayed in the workspace content area, as shown in Figure 6-1.

Note: For more information about using the Service Explorer, refer to Chapter 7, "Viewing Metadata Using the Service Explorer."

Using the General Tab

This tab displays general configuration information about the web service, such as the target namespace, SOAP version, the status of the ADO.NET control. In addition, it provides the option to select basic authorization for the web service. Figure 6-1 displays the General tab page for the ADDRESS.ws.

Figure 6-1 Native Web Service: General Tab



Note: You can set security policies for a native web service using the Basic Auth Required property. For more information, refer to Securing Native Web Services section in Chapter 5, "Securing ALDSP Resources."

Using the General tab, you can also perform the following functions:

- Test the Generated Web Service
- View the WSDL
- Export the Static JAR File

Test the Generated Web Service

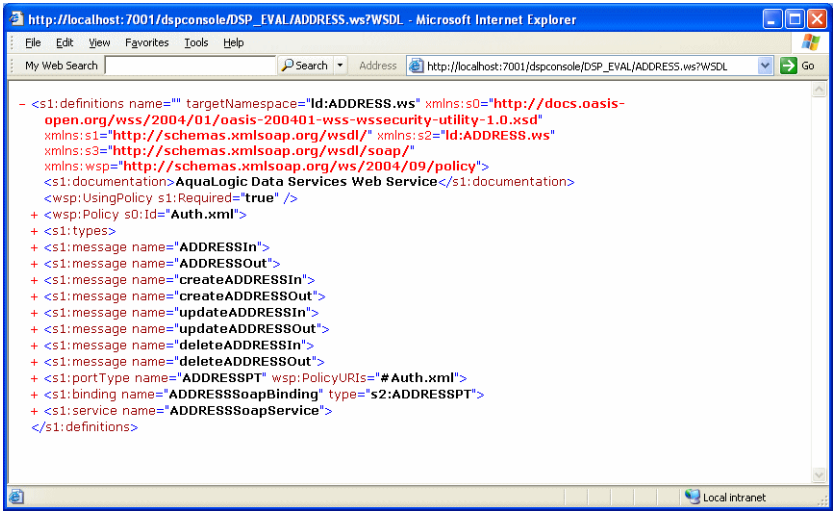
Click the Test Web Service link on the General tab. This displays the WebLogic Test Client, which allows you to test the web service as shown in Figure 6-2.

Figure 6-2 WebLogic Test Client

View the WSDL

Click the View WSDL Definition link to open the WSDL definition for the web service. A sample WSDL definition looks similar to the displayed in Figure 6-4.

Figure 6-3 WSDL Definition



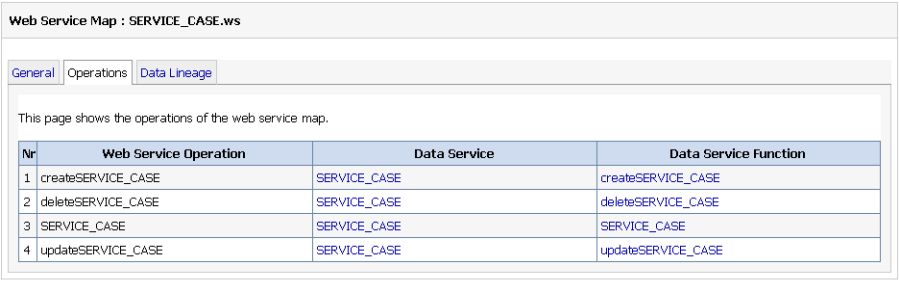
Export the Static JAR File

Click the Export Static Client Jar link (Figure 6-1) to export the web service artifacts. This option is useful when a client needs to consume the data service as a static web service.

Using the Operations Tab

This tab displays information about underlying data service and data service functions associated with the web service as shown in Figure 6-4.

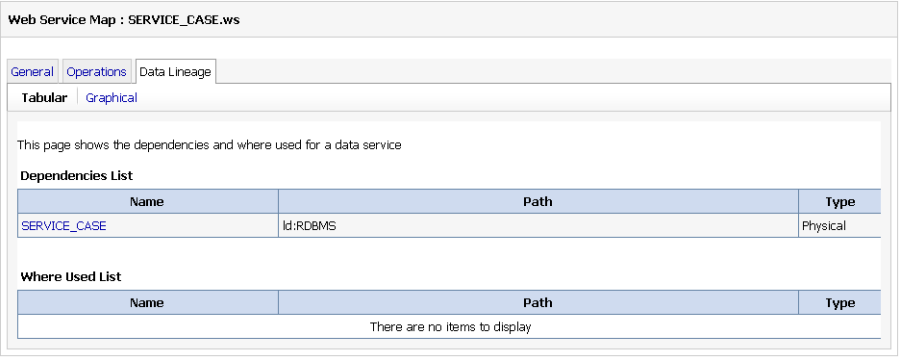
Figure 6-4 Native Web Service: Operations Tab



Using the Data Lineage Tab

This tab displays the dependencies and where used information for the web service. The information is same as the data lineage for the referenced data service as shown in Figure 6-5.

Figure 6-5 Native Web Service Data Lineage



Generating a Web Services Mediator Client JAR File

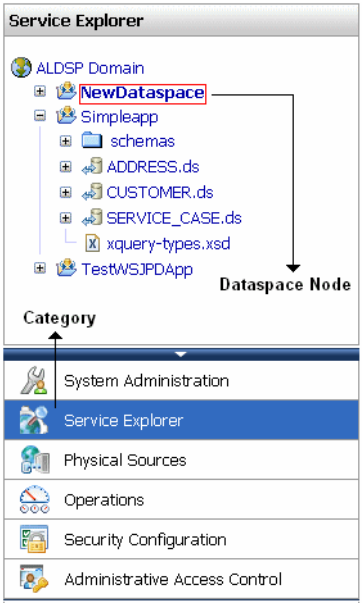
To use the Static Mediator API in a web services-enabled client application, you must generate a Web Services Mediator Client JAR file. This JAR file contains the Static Mediator API interfaces, plus all the necessary SDO-compiled schemas for a data space. This section explains how to generate a Web Services Mediator Client JAR file using the Administration Console.

Tip: For information on the Static Mediator API and on writing web services-enabled clients, see the *Client Application Developer's Guide*.

---

1. Start the ALDSP Console. See the ALDSP Administration Guide for instructions.
2. Select the Service Explorer category, as shown in Figure 6-6.

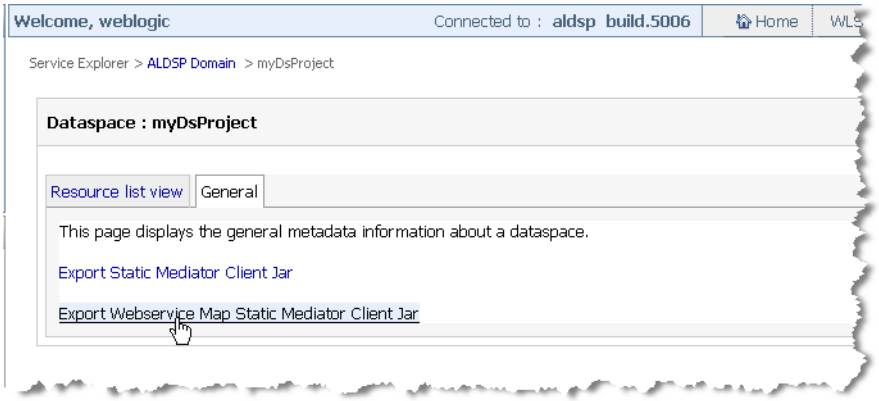
Figure 6-6 Selecting the Service Explorer Category



3. In the explorer, click the Data Space node that you wish to export. In Figure 6-6, the node is called myDataSpace.

4. In the Data Space pane, select the General tab.
5. Select Export Webservice Map Static Mediator Client Jar, as shown in Figure 6-7. The mediator JAR file is saved to your local file system.

Figure 6-7 Exporting the Client JAR File



Generating a Mediator Client JAR File

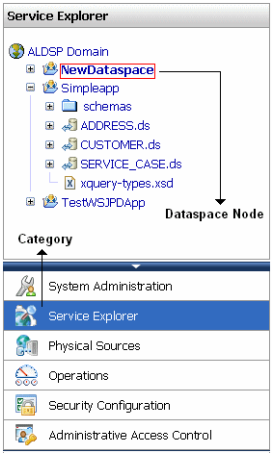
To use the Static Mediator API in a web services-enabled client application, you must generate a Mediator Client JAR file. This JAR file contains the Static Mediator API interfaces, plus all the necessary SDO-compiled schemas for a data space. This section explains how to generate a Mediator Client JAR file using the Administration Console.

Tip: For information on the Static Mediator API and on writing web services-enabled clients, see the *Client Application Developer's Guide*.

---

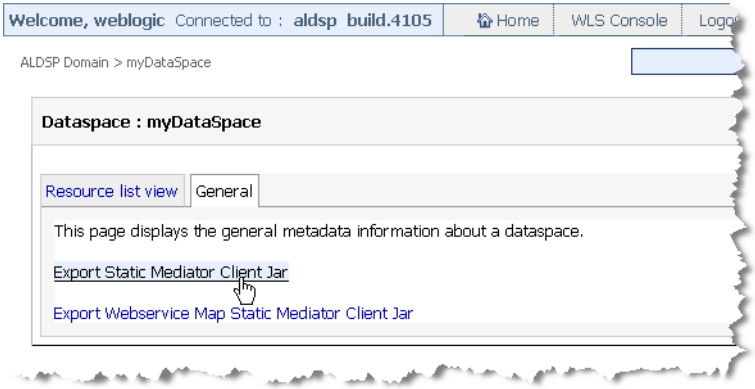
1. Start the ALDSP Console. See the ALDSP Administration Guide for instructions.
2. Select the Service Explorer category, as shown in Figure 6-8.

Figure 6-8 Selecting the Service Explorer Category



3. In the explorer, click the Data Space node that you wish to export. In Figure 6-8, the node is called myDataSpace.
4. In the Data Space pane, select the General tab.
5. Select Export Static Mediator Client Jar, as shown in Figure 6-9.

Figure 6-9 Exporting the Client JAR File



CHAPTER 7

Viewing Metadata Using the Service Explorer

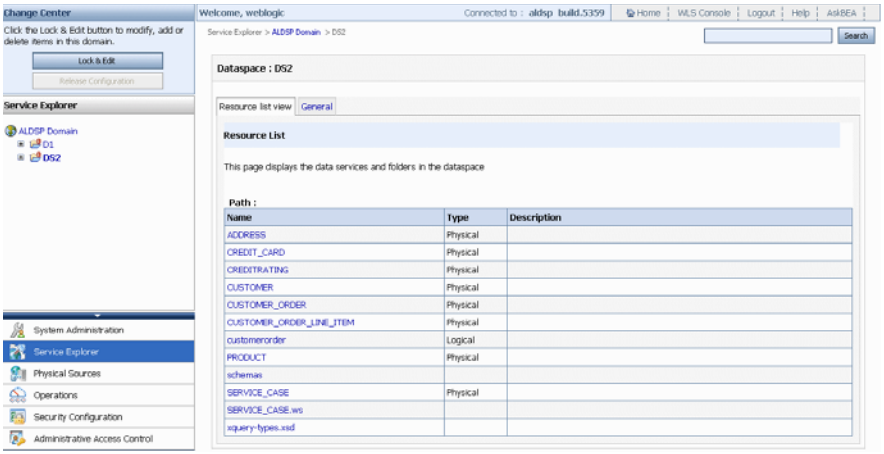
In ALDSP Administration Console, Service Explorer enables you to view metadata information on data services, their functions, and their dependencies in the active WebLogic Server.

This chapter describes how to view and analyze metadata for data services, functions, and Web services using the Service Explorer. It includes the following sections:

Introducing Service Explorer

The Service Explorer enables you to view metadata related to a data space project deployed on the server. The metadata in ALDSP includes metadata documents that the data model represents, which consist of information about the data services, their functions and return types, and dependencies between data services. Figure 7-1 displays the Service Explorer tab and the metadata for the corresponding data service in the Detail Book (right pane).

Figure 7-1 Service Explorer



ALDSP metadata is mainly used by:

- ALDSP administrators to monitor the effects of changes to underlying data sources.
- Developers of data services client applications to determine the data services that are available and their calling conventions.

Using the Service Explorer

The Service Explorer enables you to access metadata in the following ways:

- View metadata for data services. For more information, see "Analyzing and Viewing Data Services Metadata" on page 7-4.
- View metadata for data service functions. For more information, see "Viewing Data Service Functions Metadata" on page 7-10.
- View metadata for web services. For more information, see "Viewing Web Service Metadata" on page 7-14.
- Search for metadata in a data space project. You can perform basic or advanced search on metadata. For more information, see "Searching Metadata" on page 7-16.

Web Browser Requirements for Data Lineage Graph

You need to install the Adobe® SVG Viewer plugin for Internet Explorer and Netscape Web browser to view the data lineage feature. It can be downloaded from:

`http://www.adobe.com/svg/viewer/install/main.html`

Table 7-1 outlines the other web browser requirements to view the data lineage graph. If your system does not meet the requirements stated in the table, revert to the tabular view of the Service Explorer.

Table 7-1 Browser Support Information for Viewing Data Lineage Graph

Browser (Version)	SVG Viewer Information	Additional Information
Internet Explorer (6.0 and above)	Can auto-detect SVG viewer. If SVG viewer is not installed, a message is displayed with the URL to download the viewer. Install the viewer and the data lineage graph will be visible instantly.	• On Windows platform only.
Mozilla Firefox (1.5 and above)	Has native SVG viewer support.	• On Windows, Linux platforms.

Table 7-1 Browser Support Information for Viewing Data Lineage Graph (Continued)

Browser (Version)	SVG Viewer Information	Additional Information
Netscape (7.x and 8.x)	Can auto-detect SVG viewer. If SVG viewer is not installed, a message is displayed with the URL to download the viewer. Install the viewer and the data lineage graph will be visible instantly.	• Netscape 8.x is available on Windows platform only. • Netscape 7.1 is available on Windows and Linux platforms. However, the data lineage graphical view is not available. • You need to add the URL to the list of trusted sites to view the data lineage graph. Perform the following steps: 1. Click the **Open Site Controls** icon on the browser tab when you log in to the Administration Console. 2. In the pop-up dialog box, select the **I trust this site** radio button. 3. Click **Done** to save your preference. This will enable you to view the data lineage graph.
Netscape 9.0	Has native SVG viewer.	• On Windows platform only.

Analyzing and Viewing Data Services Metadata

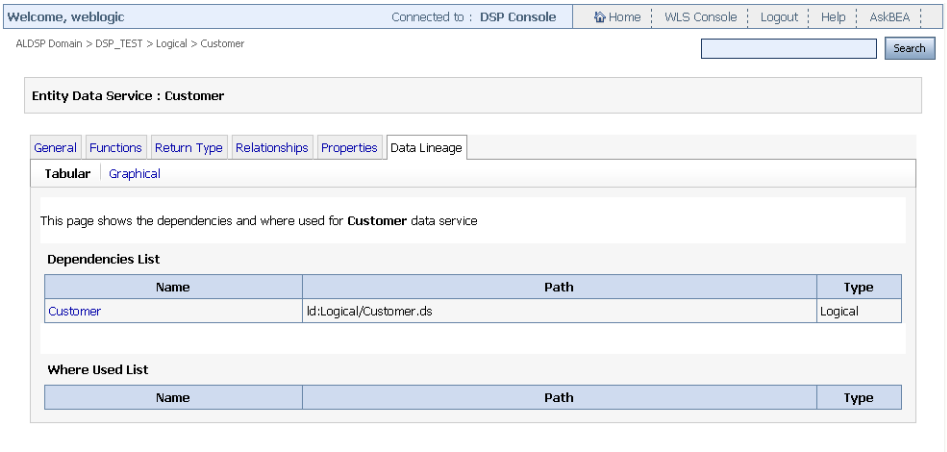
There are two kinds of data services in ALDSP, entity and library. Entity and library data services can be either physical or logical type.

- **Physical** data service**s** represent a single data source, typically a relational database table, stored procedure, or a web service.
- **Logical** data service**s** can be composed from multiple data sources and represent a view of data which is typically not available from any single data source.

The metadata that is available through the Service Explorer varies depending on whether a data service is physical or logical. Logical data services always have dependencies while the physical data services always have dependents.

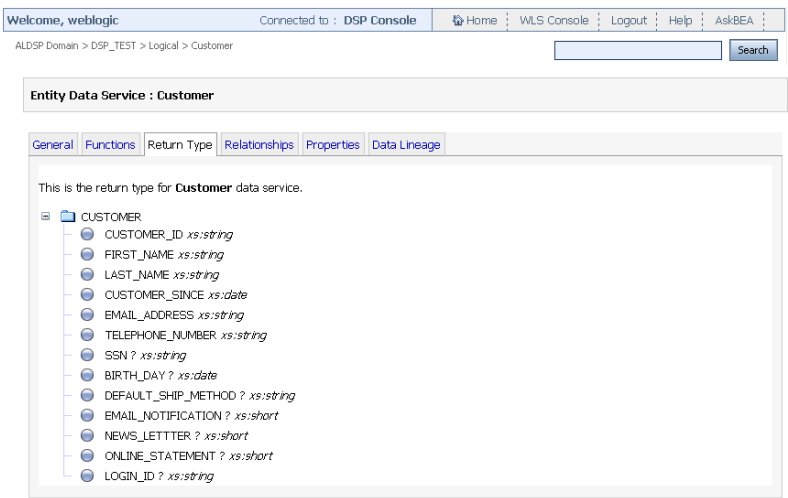
Figure 7-2 illustrates a tabular view of dependencies and the where used information of a logical data service.

Figure 7-2 Logical Data Service Dependencies and Where Used



For a logical data service, the return type displays the schema of the data from multiple data sources, according to the design of the data service, as illustrated in Figure 7-3.

Figure 7-3 Return Type for a Logical Data Service



For more information about the data service model, refer to Designing Logical Data Services in the *Data Services Developer's Guide*.

You can browse entity data service metadata including general information about a specific data service, its data lineage, its read function and return type, relationships, and dependencies.

To view data service metadata:

1. Select the Service Explorer tab in the Navigation pane.
2. Select the data service for which you need to view the metadata. By default, the General tab is displayed (Figure 7-4), which provides information such as owner, creation date, and return type for the data service.

Figure 7-4 Data Service Metadata

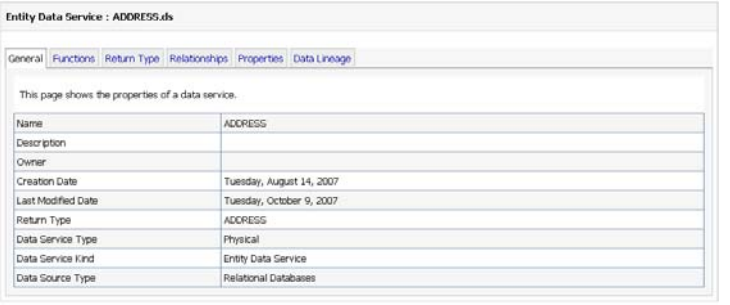


Table 7-2 describes the data service metadata information accessible through various tabs.

Table 7-2 Metadata Information

Tab	Description
General	Provides general configuration information about the data service, including the following: • **Name:** The name of the data service. • **Description:** A user-supplied description. • **Owner:** The owner of the service. • **Creation Date:** The date when the data service was created. • **Last Modified Date:** The date on which the data service was last changed. • **Return Type:** The type returned by the data service. • **Data Service Type:** Either physical or logical. For more information about data service types, see "Viewing Data Service Functions Metadata" on page 7-10. • **Data Service Kind:** Either library or entity data service. For more information about data service kinds, refer to "Viewing Data Service Functions Metadata" on page 7-10. • **Data Source Type:** The type of the data source such a relational or web service.
Functions	Displays a table of read, create, update, and delete functions. In addition, it provides the following information: • **Function Name:** Name of the function. • **Type:** Type of the function, which can be read, create, update, navigate, delete. In addition, it lists library functions and procedures also. • **Visibility**: The value can be public, protected, or private. • **isPrimary:** The value is boolean and can be either true or false. • **Parameter Types:** The parameters for each function listed in the table • **Return Type:** The return type for each function listed in the table
Return Type	Displays the content of the schema associated with the return type of the data service. This tab does not appear in case of library data services.
Relationships	Displays a table of related navigation functions. The table also lists the parameter names, if any, and return type for each function.

Table 7-2 Metadata Information (Continued)

Tab	Description
Properties	Lists any user-defined properties assigned to the data service.
Data Lineage	Provides a visual representation of the lineage between the currently selected data service. Relationships can be displayed in one of the two possible directions: • Dependencies • Where used Each entry includes name and path information. You can view data lineage in graphical or tabular views.

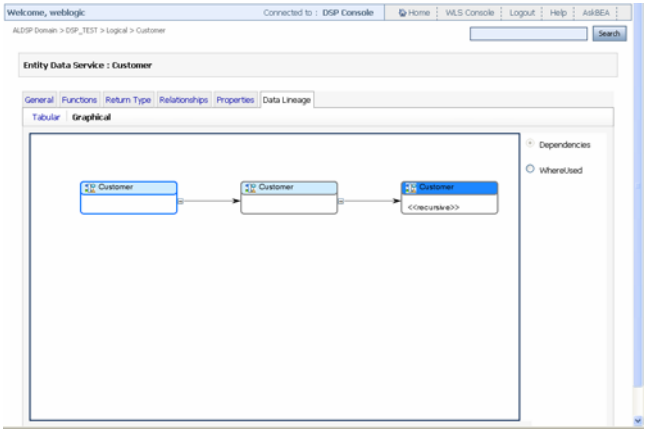
Data Service Lineages

Data service lineages can be viewed in graphical or tabular format and all kinds of data services are traceable. The graphical view is ideal for getting a visual understanding of the lineage associated with a particular data service. In the tabular view, there are two ways for viewing a data service lineage:

- **Where used view:** This view displays the currently selected data service and other data services, which use this data service. This is the *downstream view.*
- **Dependency view.** This view displays the currently selected data service and the data services it is dependent upon. This is the *upstream view.*

  In case of navigation functions, references to other data services through a navigation function are not considered as dependencies. This is because navigation functions can be created automatically during the import metadata process. For details see Creating and Updating Physical Data Services, in the *Data Services Developer's Guide*.

Figure 7-5 Customer Data Service and Its Dependents



Data Lineage Viewing Options

Once visual rendering appears, several options become available:

- **Panning (Alt + Click, then drag).** Allows you to move through the lineage representation in any direction.
- **Zoom out (Ctrl + Shift + Click).** Allows you to zoom out, providing information on data services that are further removed from your current selection.
- **Zoom in (Ctrl + Click).** Allows you to zoom in on a set of data services.
- **Expanding/Contracting.** You can use the +/- sign adjacent to the object to expand or collapse that node.

You can navigate to a new data service simply by double-clicking it in the lineage diagram.

Note: Panning and Zooming operations work only with the Adobe SVG Viewer.

Viewing Data Service Functions Metadata

You can browse metadata associated with a function.

To display function metadata:

1. Select a function in the Navigation pane.

   The console displays the General metadata associated with the function.

2. Click the corresponding tab to display general information, function dependencies, where used information, properties, and the return type.

   Figure 7-6 illustrates the function metadata displayed.

Figure 7-6 Function Metadata

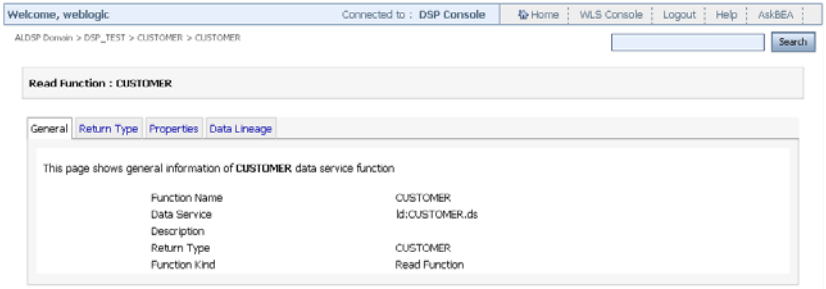


Table 7-3 describes the function metadata available.

Table 7-3 Function Metadata

Function Metadata	Description
General	General metadata information for the function, which includes the following: • **Function name:** The name of the function. • **Data Service:** The containing data service. • **Description:** A user-supplied description of the function. • **Return Type:** The type returned by the function. • **Function Kind:** The type of function such as read, create, update, delete, navigate, and library.
Return Type	Displays the content of the schema associated with the return type of the function. This tab does not appear in case of library functions.
Properties	Displays any user-defined properties associated with the function.
Data Lineage	Provides a visual representation of the relationships between the currently selected data service read, navigation, or private function. Lineage can be displayed in one of the two possible directions: • Dependencies • Where used Each entry includes name and path information.

Data Service Function Lineages

Data service function lineages can be viewed in graphical or tabular format. The graphical view includes all functions that directly or indirectly call your selected function, or are called by your selected function. In tabular view, there are two ways to view a data service function lineage:

- **Dependency view.** The currently selected data service function and any functions that it calls (said another way, it depends upon).
- **Where used view.** The currently selected data service function and any functions that make use of it (said another way, depend on it).

To view the function lineage

1. Select a data service from the Navigation pane.
2. Click the data service and then select from the list of available functions.

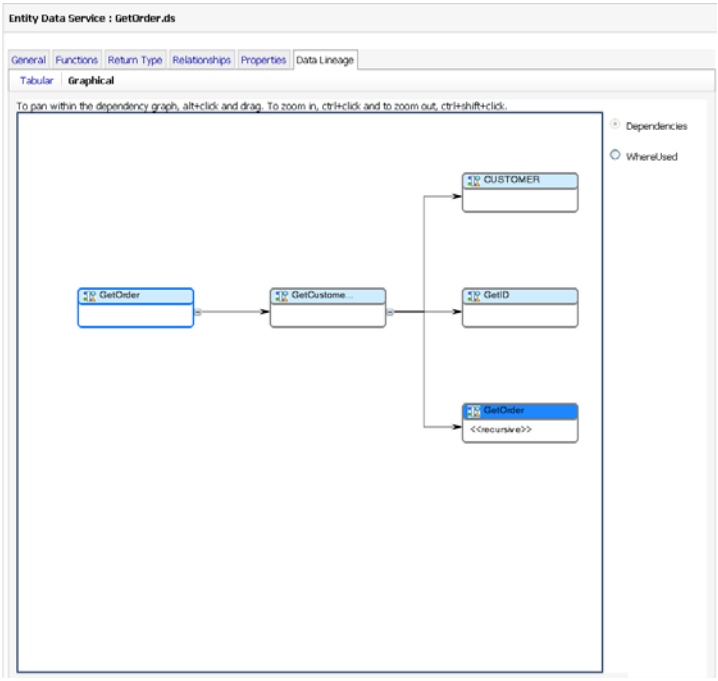
For data lineage viewing options, refer to "Data Lineage Viewing Options" on page 7-9.

Cyclic Dependency

Cyclic dependency can be observed in a graphical view of both data service lineages and data service function lineages. If a data service is used more than once, each instance of the data service in the graphical view is indicated in a dark blue color. Similarly, if a data service function is used more than once, each instance of the data service function in the graphical view is indicated in a dark blue color. Cyclic redundancy is applicable only when the duplicating nodes are part of the same branch.

Figure 7-7 shows the cyclic dependency of a data service. The text **`<<Recursive`** is specific to a data service and is displayed only in the case where a data service is used more than once in the same cycle.

Figure 7-7 Cyclic Dependency of Data Services in a Graphical View



Viewing Web Service Metadata

In ALDSP 3.0, data services can be mapped as a web service and you can view the metadata using the Service Explorer. The AquaLogic Data Services Platform Administration Console displays web service maps as artifacts in the data space. The Service Explorer shows the web service map artifacts in the navigation tree. The contents of the map artifact are shown in the General, Operations, and Data Lineage tabs, as shown in Figure 7-8. These tabs do not have any editable components and are only used for viewing and navigation.

Figure 7-8 Web Service Metadata: General Tab

Table 7-4 explains the information displayed for each of these tabs.

Table 7-4 Web Service Metadata

Web Service Metadata	Description
General	The general properties of the web service map and links to the test web page of the web service stack as well as the WSDL definition. In addition, it displays the properties of the web service, which include: • Target Namespace: The namespace defined for the web service. • SOAP Version: The current SOAP version of the web service, for example SOAP_11. • ADO.net Enabled: The status of this plug-in, which can either be true or false. • Transport Type: The protocol used for the transport. • HTTP Basic Auth Required: The status of the basic auth required property, which can be either true or false.
Operations	The Operations tab displays all operations of the web service maps and links to the underlying data service and data service functions.
Data Lineage	The Data Lineage Tab shows the data lineage to the referenced data services, which is identical to data services data lineage. You can view the data lineage in tabular as well as graphical format like data services data lineage. For more information on data services data lineage refer to "Data Service Lineages" on page 7-8.

Searching Metadata

The ALDSP console provides both basic and advanced search facility. You can use the search capabilities to locate data services based on metadata associated with the services. You can then generate a report using the results from either of the search modes.

Search algorithms that include wildcards are based on standards governing regular expression syntax. For detailed information on regular expression syntax see the following currently available Web site:

- http://en.wikipedia.org/wiki/Regular_expression

Alternatively, any other standardized regular expression reference can be consulted.

The following topics are covered in this section:

- Search Guidelines
- Performing a Basic Metadata Search
- Performing an Advanced Metadata Search
- Generating Reports

Search Guidelines

ALDSP Administration Console uses inherent Java regular expressions or **regex** patterns to implement text search. Following are the features that you can use to perform search operations on the console:

- All text entries in search boxes (basic or advanced) can have Java **regex** patterns.
- **.*** is used to map zero or more of any char values.
- **.?** is used to optionally map any char values.
- Search is case insensitive.

- Java regex pattern needs to match the entire string for a successful search. For example, if a data service name is customer, the following matches are displayed after the search is complete:
  - "*mer"
  - "cus*"
  - "customer"
  - "Customer"
  - "*to*"
  - "cus*mer"

  The following will not match
  - “cus”
  - “mer”

Note: Search patterns may be heavy for the server to process, which may cause server slowdown. Therefore, it is advised that you provide correct and specific details to make search successful and less costly. For example, an asterisk (*) in the beginning of a pattern makes the search operation less time consuming and costly than one at the end.

Performing a Basic Metadata Search

You can search for data services based on the data service name, function name, or return type.

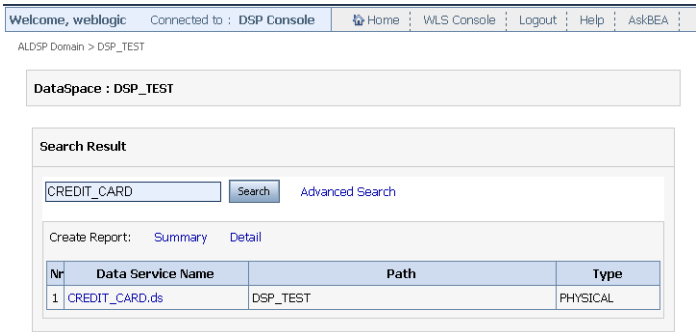
To perform a basic search enter the name of the data service, function, or return type in the **Search** box and click Search, as shown in Figure 7-9. You can also use regular expressions to search for data services. For example, to search for the CREDIT_CARD.ds, you can specify the search option as **Credit***.

Figure 7-9 Basic Search

Note: All searches are case sensitive.

Information about the corresponding data service is displayed. The information includes the data service name with links to navigate through the data service, path, and type of the data service as shown in Figure 7-10.

Figure 7-10 Basic Search Facility



3. To create a summarized or detailed report, click Summary or Detail options. For more information about generating reports, see "Generating Reports" on page 7-21.

   To perform an Advanced Search with additional search criteria, you can select the Advanced Search option. For more information, see "Performing an Advanced Metadata Search" on page 7-18.

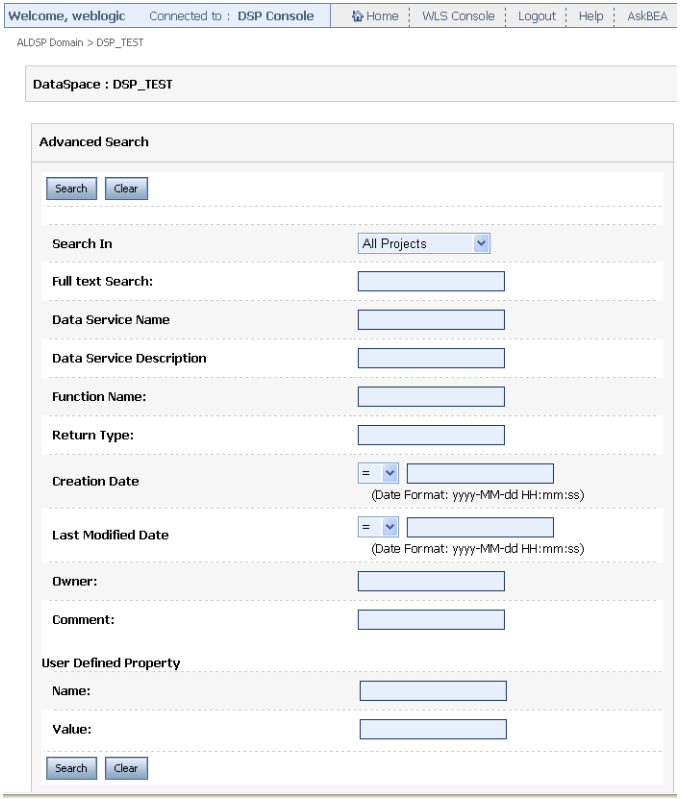
Performing an Advanced Metadata Search

You can use the advanced search facility to narrow your search criteria in cases when a basic search produces a large number of results. Using the advanced search option, you can specify criteria such as creation date, last modified date, owner, comments, and user-defined properties.

To perform an advanced search:

1. Click the Search button on the top-right corner of the console. This displays the Advanced Search screen as shown in Figure 7-11. The Search box should be empty when you click Search otherwise basic search is performed.

Figure 7-11 Advanced Search Screen



2. Enter the search criteria, as appropriate, and click Search. Table 7-5 describes the criteria you can specify using the advanced search facility.

Table 7-5 Advanced Search Criteria

Search Options	Description
Search In	The name of the folder you want to search.
Full Text Search	The equivalent of basic search, which can be combined with other advanced search criteria to get the matching results.
Data Service Name	The name of the data service.
Data Service Description	The user-supplied description of the data service.

Table 7-5 Advanced Search Criteria (Continued)

Search Options	Description
Function Name	The name of the function appearing as part of the data service.
Return Type	The return type of the data service.
Creation Date	The date the data service was created. You can select a relational operator when specifying the date from among the following: • = **(On this date).** Matches the date specified. • < **(Earlier than).** Matches dates earlier than the specified date. • <= **(On this date or earlier).** Matches the specified date or earlier dates. • >= **(On this date or later).** Matches the specified date or later dates. • > **(Later than).** Matches dates later than the specified date.
Last Modified Date	The date the data service was last modified. You can select a relational operator when specifying the date.
Owner	The owner of the data service.
Comment	The comment associated with the data service.
User Defined Property: Name	The name of a user-defined property.
User Defined Property: Value	The value associated with a user-defined property.

Note: All the search options in an advanced search can use regular expressions except the user defined properties: name and value.

The search results appear in the Search Results pane. The information displayed in the search results includes the name of the data service, the path for identifying the data service, and the of the data service, which can either physical or logical. For more information about the type of data services, refer to "Viewing Data Service Functions Metadata" on page 7-10.

Note: The information in the Search Results for basic and advanced search are the same.

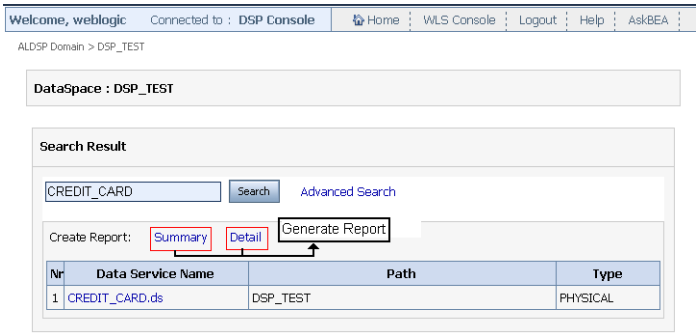
3. Click the Summary or Detail option to generate a report from the search results. For more information about generating reports, see "Generating Reports" on page 7-21.

Generating Reports

You can generate summarized or detailed reports for both basic or advanced search results. To generate a report:

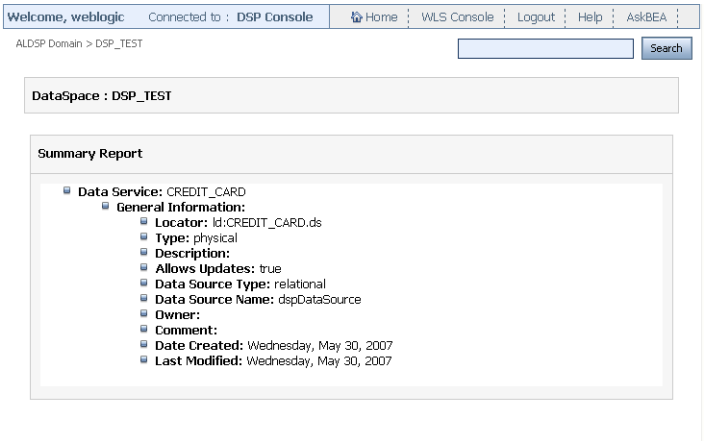
1. To generate a summarized report, click Summary from the Search Results page, as shown in Figure 7-12.

Figure 7-12 Generating Reports



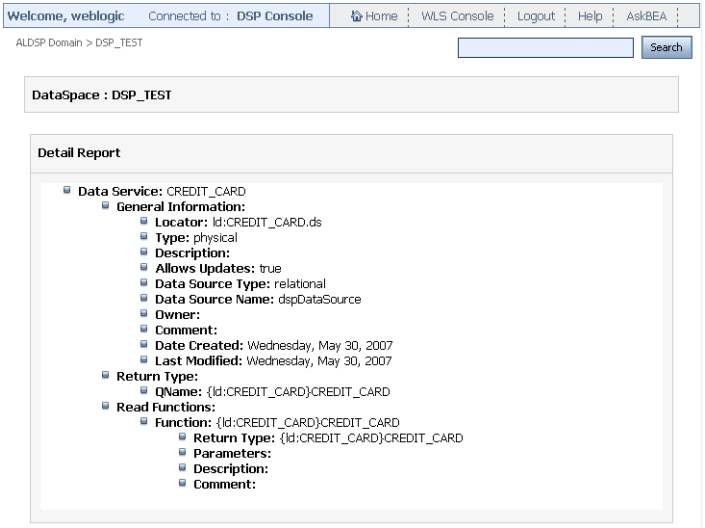
The summary report is generated as shown in Figure 7-13.

Figure 7-13 Summary Report



2. To generate a detailed report, click Detail on the Search Results page. This displays a detailed report of the data service, as shown in Figure 7-14.

Figure 7-14 Detailed Report



CHAPTER 8

Configuring Query Results Cache

This chapter describes how to set up and manage caching for data services in ALDSP. It contains the following sections:

- Understanding Results Caching
- Setting Up Caching
- Monitoring and Purging Data Cache

Note: Caching is not available for ad-hoc queries and XQuery functions for security.

Understanding Results Caching

By caching data returned by data service functions, you can improve response times for clients and reduce the processing burden on back-end systems.

Note: When results sets are cached, there are chances of using stale data instead of the updated information.

To use results caching, a database that is certified for ALDSP caching support should be installed and running. Such DBMS systems are identified in the Supported Configurations.

You can specify if you want to enable caching for functions in the Data Services Studio Overview mode. When you run the function the first time, the query results for the function are saved to a local query results cache. The next time the function is run with the same parameters, ALDSP checks the cache configuration and, if the results have not expired, retrieves the results from the cache rather than from the external source.

A cache entry exists for the results of each function invocation with distinct parameters. In cases when a cache-enabled function is invoked twice with two different parameters, two cache entries will be created.

By default caching is disabled. If you enable it, you can configure the cache and its time-to-live (TTL) for individual data service functions through the ALDSP Administration Console.

To enable caching for data service functions, you need to:

- Enable caching at the dataspace level and set the cache data source and table names.
- Enable caching of data service functions, and set the cache time-to-live (which determines how long results are stored in cache).
- Monitor and clear the cache, as required.

The TTL setting is set individually for each data service function. In general, the more dynamic the underlying data, the more frequently the cache should be set to expire.

Note: Cached data is valid until the TTL limit goes past the time at which it is cached regardless of other changes in the configuration between that time.

In some cases, caching should not be used at all. Here are two examples:

- If the data changes frequently and real-time access to it is critical cache should not be enabled. On the other hand, for functions that return static data, you can configure the results cache so that it never expires. If the cache policy expires for a particular function, ALDSP flushes the cache result automatically on the next invocation.
- Cache should never be set for functions without parameters. Every physical data service function based around a relational table, for example, falls into this category. Caching such a function can have a very negative impact of performance unless the table itself has very few records.

If an ALDSP-enabled server shutdown occurs, the contents of the results cache are retained. When the server restarts, it resumes caching as before. On first invocation of a cache-enabled function, the ALDSP-enabled server checks the results cache to determine whether the cached results for this function are valid or have expired, and then proceeds accordingly.

Caching API

ALDSP provides an API allowing client applications to bypass any existing cached results in favor of the physical data source. This API provides automatic client-side cache refresh of the affected function. For details about forcing data cache update and read-through, refer to “Forcing Data Cache Read-through and Update” in the Invoking Data Services from Java Clients chapter. *Application Developer’s Guide*:

Note: Caching is particularly effective in cases when significant processing has been applied against large data sets, producing filtered results. For optimal performance, it is recommended that you not enable caching on functions that simply return large data sets directly from a relational database data source.

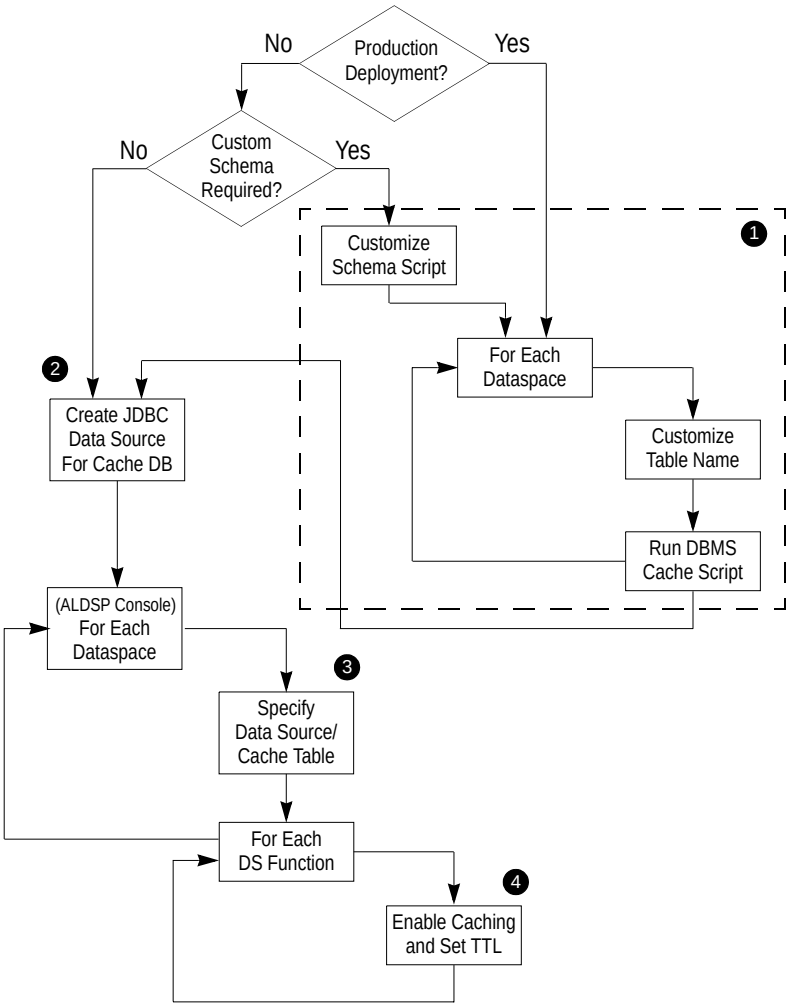
ALDSP can set up the cache table in the data source for you (if the server is in development mode), or you can create it yourself as described in the following section. Note that it is recommended that the dataspace not share cache tables. There should be separate tables for each dataspace.

Note: To prevent unauthorized access to sensitive data in the cache, it is important to maintain access control over the cache database. Also, make sure that the JDBC data source used for caching is not be used for other purposes.

Setting Up Caching

The steps for setting up cache depend on several factors, including whether you are in development or production mode and whether you need to customize the cache table schema. Figure 8-1 shows the steps for setting up caching.

Figure 8-1 Cache Setup Steps



The steps illustrated in Figure 8-1 are described in the following sections:

- Step 1: (Optional) Run the SQL Script to Create the Cache Tables
- Step 2: Create the JDBC Data Source for the Cache Database
- Step 3: Specify the Cache Data Source and Table
- Step 4: Enabling Caching by Function

Step 1: (Optional) Run the SQL Script to Create the Cache Tables

For a WebLogic server that is in development mode, you can set up the cache table automatically from the ALDSP Administration Console using the data source you choose. For production environments, or if you want to customize the cache schema, you will need to run the SQL scripts manually.

You can create the cache table using SQL scripts in the subdirectory corresponding to a particular DBMS at the following location:

`<ALDSP_HOME>/dbscripts/`

For example:

`<ALDSP_HOME>/dbscripts/oracle/dsp_cache.sql`

To create the cache table:

1. Open the script from the subdirectory that corresponds to your DBMS and modify the name of the created table so that it is unique for the dataspace.

   It is recommended that you store the cached data for each dataspace in its own cache table. For example, you can name the table <*dsname*>_CACHE.

2. Make any other schema changes, as required.

   You should not change the column names or otherwise modify the structure of the schema tables (except in specific cases, as noted in "Modifying the Cache Table Structure" on page 8-6). See Table 8-1 for information about the cache table schema.

3. Run the script.

4. Index the table based on the CHASH column (for retrieval) and the CUID column (for record updates).

   When the table is created automatically by ALDSP (as described in"Step 3: Specify the Cache Data Source and Table" on page 8-7), an index for CHASH is created. The automatically created name is the table name with "_INDEX" appended to it.

Note: On DB2, the name is truncated to a maximum of 18 characters.

Modifying the Cache Table Structure

ALDSP requires that its cache tables have a specific schema. Therefore, you should generally not modify the structure of the cache table. In some cases, however, the default column sizes may need to be adjusted based on the deployment. This may be a requirement in cases when you have data services that frequently serve result sets that are larger than the content columns in the default database tables and you are using DB2 as your DBMS.

For DB2, the scripts create the CINVKEY and CCONTENT columns (which store the results data) with a specific size, as shown in Table 8-1. If any serialized keys or content need to be larger than that size, the table schema should be adjusted accordingly before running the script.

Before attempting to implement customizations to the cache table, you should be familiar with the schema as shown in Table 8-1.

Table 8-1 Cache Table Schema

Column	Description
CUID	Unique numeric identifier for the cache entry.
CHASH	Hash value of the key (CINVKEY) as a 64-bit integer. This field enables fast searches, since searching by the key itself is inefficient as the key is stored as a binary object. (In fact, searching by the key itself is impossible for any DBMS for which the scripts create the CINVKEY as a BLOB type.)
CEXPIRE	Timestamp value indicating when the record expires. This value is computed during record insertion as current time plus the TTL value defined for the function.
CFID	Serialized name of the function. When the table is created automatically, VARCHAR(512) type is used. The value should be adjusted to a lower or higher size if names of all functions in a dataspace are smaller or if some names are larger then 512 characters.

Table 8-1 Cache Table Schema (Continued)

Column	Description
CFARITY	The number of arguments the function accepts. This is used to differentiate functions in case of function overloading (not currently used).
CINVKEY	The serialized invocation identifier consisting of the function and its arguments (created with a size of 50 kilobytes on a Pointbase DBMS).
CCONTENT	Binary data constituting the cached results. (Created with size of 1 gigabyte for DB2 and 200K for a Pointbase DBMS.)

Step 2: Create the JDBC Data Source for the Cache Database

After creating the cache table, you can use the WebLogic Administration Console to create a JDBC data source on the WebLogic Server that points to the database that you have set up for the ALDSP cache.

Note: If using Oracle as your cache database, you must set the Honor Global Transactions setting to `FALSE` (it is set to `TRUE` by default). When you create the Oracle JDBC data source in the WebLogic Administration Console, you must uncheck the Honor Global Transactions box.

Once created, you can enable the result cache as described in the following section.

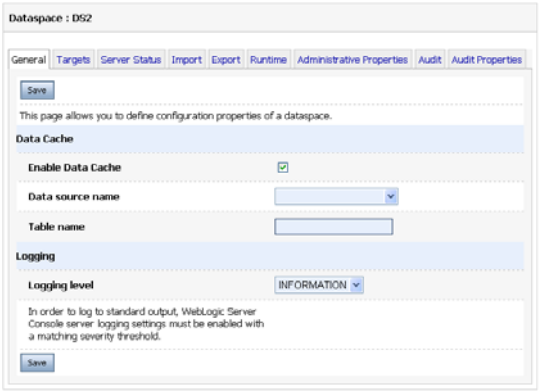
Step 3: Specify the Cache Data Source and Table

After configuring the table that you want to use for caching as a JDBC data source in the WebLogic Administration Console, you can set up the cache tables using the ALDSP Administration Console.

To specify the cache database and enable caching:

1. Select the dataspace node in the Navigation pane. The General tab appears, as shown in Figure 8-2.

Figure 8-2 Enabling Results Caching for a Dataspace



2. Click Lock & Edit to acquire the lock.
3. In the Data Cache section of the General tab, click Enable Cache.
4. Specify the JNDI name of the data source you configured for the cache table in the Data source name list box.

   If you did not create a cache table, choose the data source in which you want ALDSP to create the cache table.
5. If you created a custom cache table for the dataspace, enter its name in the Cache table name field.

   Otherwise, either enter another name for ALDSP to use when creating the table or leave the field blank, in which case the default name, <*dsName*>_CACHE, will be used.
6. Click Save > Activate Changes.

Once caching is enabled, you need to configure results caching for each function.

Step 4: Enabling Caching by Function

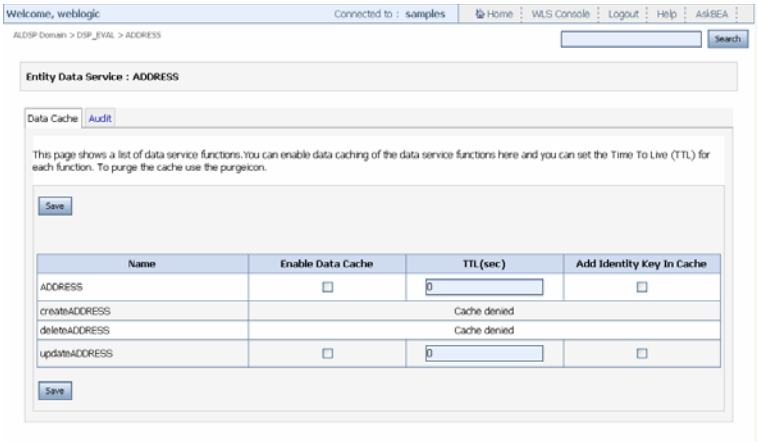
After enabling Cache settings for the dataspace, you can configure data service function caching. For each function, you can specify whether caching should be enabled, and set the time-to-live (in seconds) for cache entries.

To enable caching by function:

1. Make sure that the System Administration category is selected.
2. Click the data service name in the Navigation pane.

   The Data Cache page appears, as illustrated in Figure 8-3.

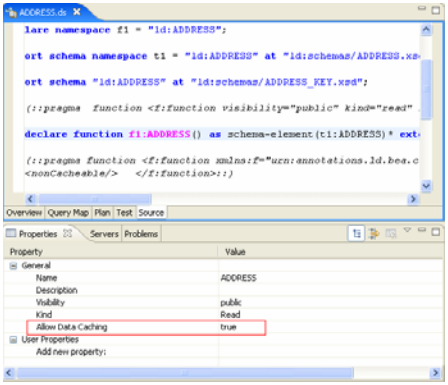
Figure 8-3 Enabling Caching by Function



3. Click Lock & Edit to acquire the lock.
4. Select the Enable Data Cache checkbox for each function for which you want to enable caching.

   Note: Make sure that you set the Allow Data Caching property for the function to true in ALDSP IDE, before enabling data caching on the console. For example, to enable caching for ADDRESS(), set Allows Data Caching property to true in ALDSP IDE, as shown in Figure 8-4.

Figure 8-4 Configuring the Allow Data Caching Property in ALDSP IDE



5. Enter a time-to-live (TTL) value, in seconds, for each cache-enabled function.

   The more dynamic the underlying data, the more frequently the cache should be set to expire.

6. Select the Add Identity Key in Cache if you want to store the caching information of the identity keys of ALDSP resources. This enables securing the data cache values that depend on other environmental variables. For more information about this feature, refer to Caching Identity Keys for Security.

7. Click Save > Activate to save your changes.

Caching Identity Keys for Security

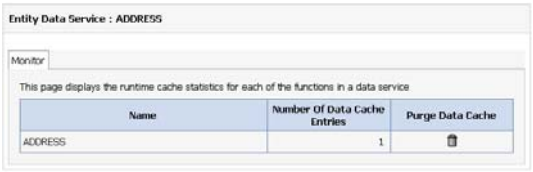
This features provides the ability to filter cached entries based on user profile. When you select the Add Identity Keys in Cache checkbox, the data cache values become user-specific, which ensures that relevant data cache entries are available to the corresponding user. For example, if two users, User A and User B, are accessing the cached values for functions, then User A will be able to view values specific to User A's transactions and User B will be able to view cached values for transactions done by User B.

This feature is especially useful when an external data source is mapped and managed through ALDSP Administration Console.

Monitoring and Purging Data Cache

You can manage function-level data caching using the Operations category. Selecting the Operation category displays the Monitor tab as shown in Figure 8-5.

Figure 8-5 Monitoring Data Cache Values



This tab provides runtime cache statistics for functions and allows you purge the cache.

The **Number of Data Cache Entries field** displays the number of results that have been cached in the data cache.

Note: The Operations category pertains to the runtime monitoring of deployed artifacts. In other words, the Operations category depends on the core (deployed) session. By contrast, other categories such as Service Explorer and Security relate to the session in progress.

Purging Data Cache

Purging the cache removes cached entries from the cache database. When the cache is purged, each function executes against its data sources until it is cached again.

ALDSP flushes the cached query result for a given stored query whenever any of the following events occur:

- The data service function is modified or deleted
- Caching is disabled on the server

ALDSP flushes the cached function result on the next invocation whenever any of the following events occur:

- The function results have expired per the cache policy
- The cache policy for a function is updated or deleted

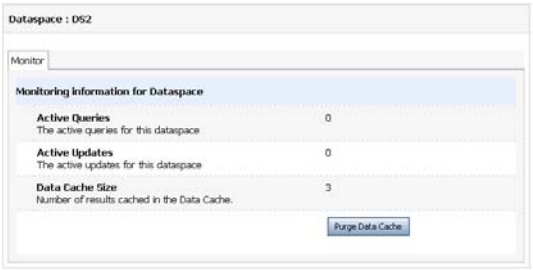
You can also purge the cache manually, either for the entire dataspace at once, or for individual functions, as described in the following sections.

Purging the Cache for a Dataspace

To purge the cache for a dataspace:

1. Select the dataspace from the navigation pane.
2. Click the Operations category.

Figure 8-6 Purging the Cache for a Dataspace



3. Click Purge Data Cache in the Monitor tab.

Purging the Cache for a Function

You can purge the cache for individual functions using the Monitor tab in the Operations category, as illustrated in Figure 8-5.

To purge cache by function:

1. From the navigation tree, select the data service for which you want to purge cache by function.
2. Click the Trash icon in the Purge Data Cache field to purge cache for the function.

CHAPTER 9

Working With Audit and Log Information

This chapter describes the auditing framework, performance profiling, and logging capabilities provided with the AquaLogic Data Services Platform. It contains the following sections:

Auditing

The auditing framework is used to collect auxiliary runtime data using a normal XQuery operation in an ALDSP dataspace. This information may be used for security auditing, performance profiling, and other purposes.

Audit Data Structure

The data structure comprises a sequence of audit records containing an unordered collection of audit properties. Each audit record contains properties of a specific type, usually identified using a hierarchal name. Each audit record corresponds to an operation performed by ALDSP. For example, access to a relational data source may generate a record of "evaluation/wrappers/relational" type that includes the following audit properties: sql, datasource, returnedRows, evaluationTime, parameters, message, and exception.

Any individual property may be configured to be collected. Each property has an individual intrinsic severity level that can be used to configure an overall threshold of what properties to collect. In certain cases, like when an exception occurs, some properties may be added to the record even if they are not configured to be collected. Typically, this information would be identifiers for a failed data source or update operation.

On the other hand, a property configured for collection may not be collected. This might be attributed to one of the following reasons:

- Data might be unavailable due to internal implementation logic.
- A property is collected by an audit based on the need to record internal conditions, for external analysis.
- If an exception is encountered. This will result in an alternate execution path and impact the information being collected.

Collected elements of the data structure can be individually configured to be:

- Submitted to the WebLogic Server auditing framework and processed by an auditing provider.
- Written to an application server or system logging stream.
- Transferred to a client application.

Note: Auditing occurs whenever the engine is invoked and the Auditing option is enabled. Timestamps and other collected data enable you to match auditing information with particular query operations.

Use the System Administration category in ALDSP Administration Console to configure audits such as setting the global audit severity level and overriding audit settings for individual properties that you may need to monitor.

Setting Global Audit Properties

There are some global auditing options that inherently apply to every aspect of the auditing process. To set these properties:

1. Acquire the lock.
2. Select the System Administration category and then the Audit tab shown in Figure 9-1, which allows you to configure these options.

By default, the audit report generation utility is turned off. Before you start generating audit reports, you need to enable auditing.

Note: With auditing enabled, performance may be affected, depending on the audit levels and the number of properties being audited.

Figure 9-1 Audit Options

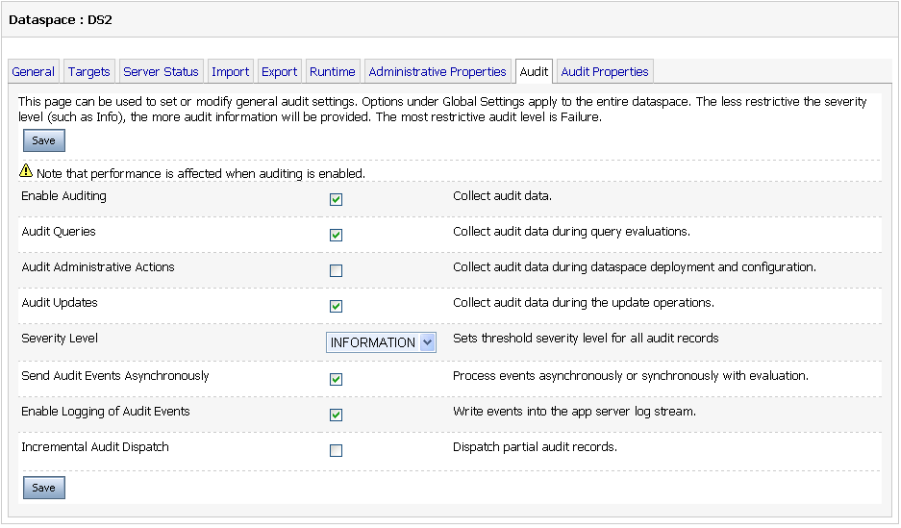


Table 9-1 describes available global auditing options. Select the respective check box in the ALDSP Console to implement the required audit options.

Table 9-1 ALDSP Global Auditing Options

Options	Description
Enable Auditing	Determines whether the auditing is activated or not. **Note:** When auditing is enabled, performance can be affected to a degree, depending on the audit level and the number of items being tracked.
Audit Queries	Determines whether the auditing is activated or not, during a query evaluation.
Audit Administrative Actions	Collects audit data during dataspace deployment and configuration.
Audit Updates	Determines whether auditing is activated or not during update operations.
Severity Level	Determines the level of information to be captured by the auditing process. See Auditing Severity Levels section for more information.
Send Audit Events Asynchronously	Determines whether the events are processed synchronously or asynchronously.
Enable Logging of Audit Events	Determines whether the auditing information is to be included in the application server log file. **Note:** If you enable this option (logging), ensure that the Log Level value in the General tab is set to either Info or Debug. Any other value will result in the log file not accepting any information.
Incremental Audit Dispatch	Determines if the audit records are to be dispatched partially or completely every time there is a new record.

Auditing Severity Levels

You can set the severity levels using the Severity Level drop down list in the Audit tab (Figure 9-1). Severity levels are similar to those provided with WebLogic Server security. For WebLogic Server details, see Message Severity.

Table 9-2 ALDSP Audit Severity Levels

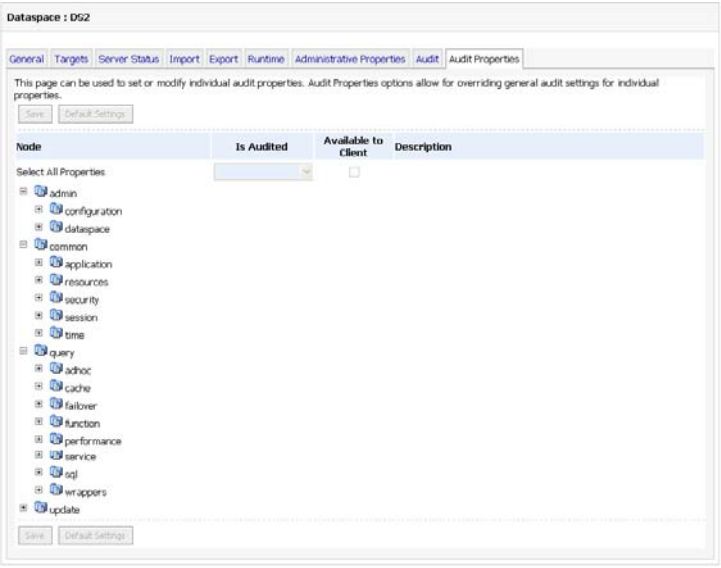
Level	Description
Debug	This setting is often referred to as “verbose”. Any audit property that can be added to the audit report is collected.
Information	Properties with information or higher conditions are collected for the audit report.
Warning	Properties with warning or higher conditions are collected for the audit report.
Failure	Properties with error or more higher conditions are collected for the audit report.

Setting Individual Auditing Properties

This section describes the individual auditing properties that you can audit and to what level. To configure these auditing properties:

1. Acquire the lock.
2. Select the System Administration category
3. Click the Audit Properties tab as shown in Figure 9-2.

Figure 9-2 Audit Properties Tab



4. After configuring audit properties, click Save > Activate Changes to implement the audit settings.

Note: Click Default Settings to rearrange the auditing properties to the default values.

Audit properties can be configured at different levels and you can select the level using the Is Audited drop-down list. Table 9-3 lists the audit levels that you can set for each property. All levels listed in the table are not applicable to all the properties. Typically, each property has only three levels to choose from.

Note: If you want the property-specific audit information to be returned to a client, then select the Available to Client check box.

Table 9-3 Setting Individual Audit Properties

Level	Description
Always	In this setting, the audit information of the property is always collected.
Never	In this setting, the audit information of the property is always ignored.

Table 9-3 Setting Individual Audit Properties (Continued)

Level	Description
At Default Level	This setting configures the property at the default level. **Note:** This option is available only for the Select All Properties audit property.
At Info Level	In this setting, the audit information is collected if the global threshold level is **Information** or lower.
At Warning Level	In this setting, the audit information is collected if the global threshold level is **Warning** or lower.
At Failure Level	In this setting, the audit information is collected if the global threshold level is **Failure** or lower.
At Debug Level	In this setting, the audit information is collected if the global threshold level is **Debug**.

Note: After you set and apply individual auditing property settings, any changes you make on the individual properties will override the initial settings for that property only.

ALDSP Administration Console provides you with the option to select all audit properties to be audited using the Select All Properties node. You can set this property at the following levels:

- Always
- At Default Level
- Never

All other individual properties are categorized into the following overall types depending on the corresponding operation that generates the audit data:

- Admin Audit Properties
- Common Audit Properties
- Query Audit Properties
- Update Audit Properties

Admin Audit Properties

The audit information in this section pertains to the information exchanged while performing administration tasks such as configuration and application deployment. Only changes to the application made in the ALDSP Administration Console are collected during audit.

Table 9-4 Administrator Properties

Property	Description
Configuration	

notification	Records notification of deployed access control resource. For example:
	`notification: jmx.attribute.change`
	`property: MAXNUMBEROFQUERYPLANCACHED`
	`value: 101`

property	Records any instance of the property that was changed in the ALDSP Console. For example: `notification: jmx.attribute.change`
value	Records a new value instance, for example: `value: 101`
Dataspace	**Note:** This information is displayed in the audit log by default. You cannot change the audit level for this property.
name	Records the name of the dataspace
operation	Records create, modify, delete operations for a dataspace
updatediff	Records changes from the last configuration update.

Common Audit Properties

The common audit information provides the generic transaction related information. It includes generic information on the event, such as event type, application name, user id, user access rights, date, and time.

Table 9-5 Common Properties

Property	Description
Application	
eventkind	Records the type of event or operation, it could be a query or an update and so on. For example: `eventkind: evaluation`
exception	Records the exception message, if one occurred. For example: `exception: ld:DataServices/ApparelDB/CUSTOMER_ORDER_LINE_ITEM.ds, line 77, column 7: {err}FORG0005: expected exactly one item, got 0 items`
name	Records the deployed application name. For example: `name: RTLApp`

principals	Records the groups to which the user belongs. For example:
	`principals:`
	`  weblogic`
	`  Administrators`
	`  IntegrationAdministrators`
	`  PortalSystemAdministrators`

server	Records the application server's unique id. For example: `server: cgServer`
transactionid	Records the unique transaction id for the event or operation.
user	Records the user id, for example: `user: weblogic`
Resources **tempfile**	

Table 9-5 Common Properties (Continued)

Property	Description
createtime	Records the time and date when the file was created.
deletetime	Records the time and date when the file was deleted.
file	Records the name of the temporary file where the data is stored.
size	Records the size of the temporary file (in bytes), before it is deleted.
source	Records information about the operator because of which the data was spilled.
Security **Access**	
decision	Records the security access settings for the application, for example: `decision: PERMIT`
resource	Records the request for resource identifier. For example: `resource: <ld type="function"><app>RTLApp</app><ds>ld:DataServices/CustomerDB/ADDRESS.ds</ds><res>{ld:DataServices/CustomerDB/ADDRESS}ADDRESS:0</res></ld>`
resourcetype	Records the type of resource used, such as dataservice, application, submit and so on. For example: `resourcetype: function`
Session **query** **invocation**	
blocksize	Records the size of the returned serialized data block, in bytes
duration	Records the duration or the time required to compute the next block of the result, in milliseconds.
time	Records the time of call for the next data block.

Table 9-5 Common Properties (Continued)

Property	Description
Session	
SQL	
Invocation	
time	Records the date and time of the call to the next () method on the server side of the JDBC driver.
duration	Records the duration or the time required to compute the next block of the result, in milliseconds.
blocksize	Records the size of the returned serialized data block, in bytes.
Time	
duration	Records the time used to complete the audit event, in milliseconds. Calculates the time difference from initiation of the audit to its completion. For example: `duration: 2834`
timestamp	Records the time when the audit event was initiated, for example: `timestamp: Tue Feb 14 09:21:02 IST 2006`

Query Audit Properties

The audit information in this section pertains to all the information collected during query evaluation. The information includes the query itself, its result, the execution time, and details on the data source queried.

Note: When using the streaming APIs, or when using the `RequestConfig.OUTPUT_FILENAME` feature, the results of the query are not audited because they are presumed to be very large. This means the AuditEvent dispatched to the audit provider, as well as the DataServiceAudit returned to the client, will not contain a value for the audit property Query/Service/results.

Table 9-6 Query Properties

Property	Description
Adhoc	
query	Records the query that was executed.
result	Records the results obtained after execution of the query.
variablenames	Records names of the variables passed to the query.
variables	Records the external parameters or variables passed to the query.
Cache	
Data	
forcedrefresh	Boolean value where TRUE indicates the data is from a current data source or FALSE if it is from a cache.
functionid	Records the name of the function.
remainttl	Indicates the time remaining, in seconds, before the query cache is refreshed.
retrieved	Indicates whether the data was obtained from the query cache or not.
time	Indicates the duration of the cache retrieval operation.
Queryplan	**Note:** Queryplan audit properties are not collected when a function is executed from the **Test** view. This is because the function cache is not utilized for functions executed in the **Test** view.
flushed	True and set when the query plan was flushed
found	Indicates whether the query plan cache has been located or not.
inserted	Indicates whether the query plan cache has been inserted or not.
type	Indicates the type of the query plan such as XQUERY_PLAN_CACHE, SQL_PLAN_CACHE, or STORED_PROC_CACHE.
Failover	
exception	In the event of a failover, this records the exception that caused it.

Table 9-6 Query Properties (Continued)

Property	Description
function	Records the function name which can be either `fn:bea:timeout` or `fn:bea:fail-over`. For example: `function: {http://www.bea.com/xquery/xquery-fncts}timeout-with-lbl`
label	Records the user-defined label, if any. For example: `label: lab`
sourcecolumn	Records the source column of the function call. For example: `sourcecolumn: 2`
sourcefile	Records the source file of the function call. For example: `sourcefile: [ad-hoc]`
sourceline	Records the source line of the function call. For example: `sourceline: 4`
timeout	Records the time-out that was exceeded, if applicable. For example: `timeout: 0`
Function	**Note:** Function audit properties are collected only when the individual functions of a data service are selected for auditing. See Function-level Auditing for more information.
name	Records the name of the audited function. For example: `name: {ld:DataServices/CustomerDB/CUSTOMER}getCustomer`
parameters	Records the parameters passed through the audited function. For example: `parameters: CUSTOMER1`
result	Records the result after executing the audited function. For example: `result: <ns0:CUSTOMER`
Performance	
compiletime	Records the query compilation time, in milliseconds. For example: `compiletime: 19`

Table 9-6 Query Properties (Continued)

Property	Description
evaltime	Records the query evaluation time, in milliseconds. For example: `evaltime: 90`
Service	
arity	Records the number of arguments for the invoked function.
dataservice	Records the name of the data service, for example: `dataservice: ld:DataServices/RTLServices/ApplOrder.ds`
function	Records the function name of the data service, for example: `function: getCustomer`

parameters	Records the parameters passed through the query, for example:
	`parameters:`
	`1`
	`foo`

query	Records the complete text of the executed query on the data service, for example:
	`query:`
	`import schema namespace t1 = "urn:retailerType" at "ld:DataServices/RTLServices/schemas/ApplOrder.xsd";`
	`declare namespace ns0="ld:DataServices/RTLServices/ApplOrder";`

result	Records the results of the executed query, for example:
	`ORDER_10_0`
	`CUSTOMER0`
	`2001-10-01`
	`GROUND`

SQL **Procedure**	
name	Records the name of the SQL procedure.
parameters	Records the parameters associated with the SQL procedure.
parametertypes	Records the types of the parameters.

Table 9-6 Query Properties (Continued)

Property	Description
Statement	
parameters	Records the parameters of the query.
parametertypes	Records the parameter types of the query.
query	Records the text of the query.
Wrappers	
File	
exception	Records an exception, if any, when a function invoked belongs to a data service created over a File data source. For example: `exception: com.bea.ld.wrappers.df.exceptions.DFException: {bea-err}DF0004: [ld:DataServices/Demo/Valuation.csv]: Expected end of line at (row:2, column:3).`
name	Records the unique function name. For example: `name: ld:DataServices/Demo/Valuation.csv`
time	Records the time taken to query, in milliseconds. For example: `time: 20000`
Java	
exception	Records an exception, if any, when a function invoked belongs to a data service created over a Java class. For example: `exception: {ld:DataServices/Demo/Java/Physical/PRODUCTS}getFirstProduct:0, line 4, column 5: {bea-err}JFW0401: Class or Method not found exception : {ld:DataServices/Demo/Java/Physical/PRODUCTS}getFirstProduct`
name	Records the name of the service. It is always recorded if an exception property was added. For example: `name: public static int Demo.Java.JavaSource4West.echoInt(int)`

Table 9-6 Query Properties (Continued)

Property	Description
parameters	Records the external parameters passed to the service. For example: `parameters: 11`
result	Records the results of the executed query. For example: `result: 11`
time	Records the time taken to execute the query, in milliseconds. For example: `time: 20000`
Procedure	
datasource	Records the name of the data source, for example: `datasource: newDS`
exception	Records an exception, if any, when a function invoked belongs to a data service created over a stored procedure. For example: `exception: weblogic.xml.query.exceptions.XQueryDynException: {err}XP0021: "-ss": can not cast to {http://www.w3.org/2001/XMLSchema}decimal}`
name	Records the procedure identifier. It is always recorded if an exception property was added. For example: `name: WIRELESS.SIDEEFFECT_REG_PACKAGE.READ2`
parameters	Records the external parameters passed to the data service method. For example: `parameters: s 2.2 22.0 ss`
rows	Records the number of rows returned after execution of the procedure, for example: `rows: 0`
time	Records the time taken to execute the procedure, in milliseconds. For example: `time: 170`
Relational	
basesql	Records the base SQL statement text.

Table 9-6 Query Properties (Continued)

Property	Description
exception	Records the relational database query exception, if any. For example: `exception: com.bea.ld.wrappers.rdb.exceptions.RDBWrapperException:...`
parameters	Records the external parameters passed through to the data service method, for example: `parameters: ORDER_10_0 ORDER_10_1`
rows	Records the number of rows returned from the relational database, for example: `rows: 60`
source	Records the database source name. It is always recorded if an exception property was added. For example: `source: cgDataSource1`
sql	Records the SQL statement used for the query, for example: `sql: SELECT '1' AS c15, t2."LINE_ID" AS c16, t2. FROM "RTLAPPLOMS"."CUSTOMER_ORDER_LINE_ITEM" t2 WHERE ((? = t2."ORDER_ID") OR (? = t2."ORDER_ID")`
substitutionname	Records the name of the substituted SQL, if used.
time	Records the time spent executing the query, in milliseconds. For example: `time: 5000`
WS	
exception	Records an exception, if any, when a function invoked belongs to a data service created over a web service. For example: `exception: {bea-err}WSW0101: Unable to create Call : {ld:DataServices/ElectronicsWS/getCustomerOrderResponse}getCustomerOrder`
operation	Records the data service method that is executed. For example: `operation: getCustomerOrder`

Table 9-6 Query Properties (Continued)

Property	Description
parameters	Records the parameters passed through to the data service method. For example: `parameters: <ns0:getCustomerOrder xmlns:ns0="http://www.openuri.org/">`
result	Records the result returned after the query is executed. For example: `result: <ns:getCustomerOrderResponse xmlns:ns="http://www.openuri.org/"> <CustOrders xmlns="http://temp.openuri.org/SampleApp/CustOrder.xsd "> <ORDER> <ORDER_ID>ORDER_1_0</ORDER_ID> <CUSTOMER_ID>CUSTOMER1</CUSTOMER_ID>`
time	Records the time spent executing the query, in milliseconds. For example: `time: 50000`
wsdl	Records the web service description. For example: `wsdl: http://localhost:7001/ElWS/cntrls/ElDBTest.jws?WSDL`

Update Audit Properties

The audit information in this section pertains to all the information related to performing an update function. It includes information on the time taken to update the source, when it was started, the unique transaction id and so on.

Table 9-7 Update Properties

Property	Description
Error Fault	
exception	Records the value of the tostring() of the update exception.
exceptionobject	Records the exception object for dataspace audit update error.

Table 9-7 Update Properties

Property	Description
status	Records the status of the update.
updateid	Records the globally-unique update identifier.
Error	
Procedure	
arity	The arity of the update procedure.
dataservice	The data service of the update procedure.
id	The index of the update procedure invocation.
name	The name of the update procedure.
parameters	The parameters of the update procedure invocation.
result	The result of the update procedure invocation.
status	Status of the procedure executed by this update.
xid	The xid of the update procedure invocation.
Extension	
id	Records the id of the source being updated.
time	Records the time spent, in milliseconds, for the update.
Procedure	
name	Records the name of the audit procedure.
parameters	Records the parameters passed to the audited procedure.
result	Records the results of update procedure execution.
Relational	
exception	Records the update exception, if any.
parameters	Records the parameters passed during the update of the relational database.

Table 9-7 Update Properties

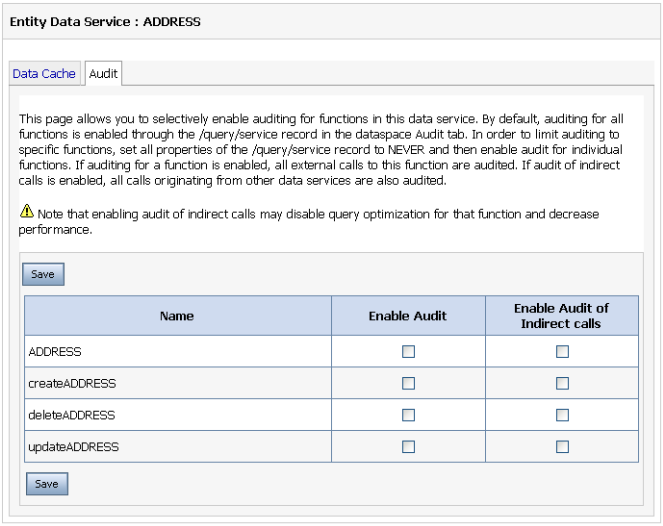
Property	Description
rowsmodified	Records the number of rows updated in the relational database, on successful completion.
source	Records the data source name. It is always recorded if an exception property was added.
sql	Records the SQL statement used during the update of the relational database.
time	Records the time spend, in milliseconds, in updating the relational database.
Service	
arity	Records the number of arguments associated with the invoked function.
dataservice	Records the data service used for the update.
parameters	Records the parameters passed to the update procedure.
procedure	Records the data service fully qualified procedure name.
result	Records the results of the update.
script	Records the complete text of the executed script.
sdocount	Records the number of top level SDOs that were submitted for the update.
time	Records the total execution time, in milliseconds, for the update.

Function-level Auditing

By default, auditing for all directly invoked functions can be enabled through the /query/service record for the dataspace using the Audit tab. However, to limit auditing to specific functions, set all properties of the /query/service record to **NEVER** and then enable audit for individual functions. To do so:

1. Acquire the lock and select the System Administration category.
2. Navigate to the data service level.
3. Select the Audit tab as shown in Figure 9-3.

Figure 9-3 Enabling Auditing for Individual Functions



If auditing for a function is enabled, all external calls to this function are audited. If the **Enable Audit of Indirect Calls** check box is selected, all calls originating from other data services are also audited.

Note: Enabling audit of indirect calls may disable query optimization for that function, and decrease performance.

Retrieving Audit Information

You can record the audit information collected in the following ways.

- **WebLogic Server Security Framework.** Each audit event is by default reported to the WebLogic Server Security Framework.
- **ALDSP Client API.** You can create a ALDSP client API to record the information collected during audit.
- **ALDSP Performance Profiling.** You can use the ALDSP audit provider for performance profiling by recording audit events generated by a dataspace.

Values of the audit properties are represented as Java objects of types: String, Integer, java.util.Date, Boolean, or String [ ].

WebLogic Server Security Framework

Each audit event is sent to the WebLogic Server Security Framework as an instance of the `weblogic.security.spi.AuditEvent` interface. Table 9-8 describes each event.

Table 9-8 WebLogic Server Audit Events

Event	Description
`getEventType()`	Returns the event type, in this case `DSPaudit`.
`getFailureException()`	Returns the exception type, if one is encountered.
`getSeverity()`	Returns the event severity level.
`toString()`	Returns the audit event details in an XML formatted representation.

Depending on the configuration, each event can be sent to the WebLogic Server audit API asynchronously and buffered by the ALDSP application.

The weblogic.security.spi.AuditEvent interface is implemented in the ld.server.audit.DSPAuditEvent interface, which collects all the information in the form of a list, where each entry is an instance of com.bea.dsp.DSPAuditEvent.

DSPAuditEvent adds the interface described in Table 9-9.

Table 9-9 AquaLogic Data Services Platform AuditEvent API

AuditEvent API	Description
`getAllRecords()`	Returns all records as a list of `com.bea.ld.DSPAuditRecord`.
`getRecords(String recordType)`	Returns all records of a particular type as a list of `com.bea.ld.DSPAuditRecord`.
`getProperty(String propertyId)`	Returns all values for a particular property, across multiple records.
`getApplication()`	Returns the ALDSP application identifier.
`getUser()`	Returns the user name of the application server user.
`getTimeStamp()`	Returns the time when the event was created.

Table 9-9 AquaLogic Data Services Platform AuditEvent API

AuditEvent API	Description
`getEventKind()`	Returns the event type, which can be `EVALUATION_EVENT`, `CONFIGURATION_EVENT` or `UPDATE_EVENT`.
`getVersion()`	Returns the event version, for example `2.1` for the ALDSP 2.1 release.

com.bea.ld.DSPAuditRecord has the interface shown in Table 9-10.

Table 9-10 ALDSP AuditRecord API

AuditRecord API	Description
getRecordType()	Returns the type of record, for example common/time/duration.
getAuditProperties()	Returns all properties in the record. Maps from String identifier to Object value.

A sample security services audit provider is included that demonstrates use of this API.

ALDSP Client API

You can use the com.bea.ld.DataServiceAudit client side instance as part of the com.bea.dsp.RequestConfig class, to collect the audit information from the client API. This class collects the audit information and returns it when the operation is successful. If the operation fails for any reason, the com.bea.ld.QueryException class can be used to collect the information as part of the exception thrown.

Note: When using Streaming APIs, auditing will not be complete until the returned XMLInputStream has its close( ) method called. This means that the AuditEvent will not be dispatched to the audit provider by the server, and the RequestConfig.getDataServiceAudit() method will return null, until close( ) is called.

Following are the four steps, with code examples, that need to be performed in order to retrieve audit information.

Initializing the RequestConfig Class

You need to initialize the RequestConfig class as shown in the following code example:

```
RequestConfig requestCfg = new RequestConfig();
requestCfg.enableFeature(RequestConfig.RETURN_DATA_SERVICE_AUDIT);
requestCfg.enableFeature(RequestConfig.RETURN_AUDIT_PROPERTIES);
requestCfg.setStringArrayAttribute(RequestConfig.RETURN_AUDIT_PROPERTIES,
new String[]
{"query/service/dataservice"});
```

Passing the RequestConfig Object

You need to pass the RequestConfig object to the invoked operation. The code example below uses getCustomer as the invoked operation.

```
CUSTOMERDocument [] custDocRoot1 = (CUSTOMERDocument
[])custDS.invoke("getCustomer", params, requestCfg);
```

Filtering Audit Data

You need to filter the data and ensure there is no unsecured access to it. Only those audit properties that are configured in the AquaLogic Data Services Platform Administration Console to be allowed to return to the client, will be returned to the client application.

Retrieving Data Service Audit

You need to retrieve the data service audit from the RequestConfig object, as shown in the code example below:

```
DataServiceAudit query = requestCfg.retrieveDataServiceAudit();
```

Retrieving Audit Properties

RequestConfig.RETURN_AUDIT_PROPERTIES is an array of string identifiers for audit properties. If you set this request attribute those specified properties will be collected for this particular evaluation even if they are not configured to be collected through the administration console. They will be returned only if it is allowed. If the RETURN_DATA_SERVICE_AUDIT request attribute is not enabled, only those properties will be returned.

RequestConfig.RETURN_DATA_SERVICE_AUDIT configures all collected audit information (that is allowed to be returned to the client application) to be returned.

ALDSP Performance Profiling

Performance profiling allows you to store select audit information in a relational database. Relational databases supported by the ALDSP audit provider are: Oracle, DB2, PointBase, Sybase, and MS SQL.

Information about audit events are stored as records in a table. A table can be used to record audit events for ALDSP dataspaces running on a server, or for dataspaces running on shared servers in a cluster.

You can deploy the ALDSP audit provider for performance profiling using the WebLogic Administration Console and configure it using the ALDSP Profiler MBean. Configuration parameters you need to set at the time of deployment are described in Table 9-11.

Table 9-11 Configuration Parameters for Performance Profiling

Parameter	Description
Data Source	Name of the JDBC data source.
Table	Name of the table in the JDBC data source that logs query execution information.
Source Table	Name of the table in the JDBC data source that logs source access information.
Summary Table	Name of the table in the JDBC data source that logs aggregated information (summary).
Event Buffer	Size of the internal event buffer. Determines the number of events a buffer stores before the profiler starts processing events.
Collect Execution Aggregate	Stores aggregates (by function) of individual query executions in memory; eventually writes the aggregate to the database.
Aggregate Group Size	Number of events processed by the profiler before the aggregates are written to the database. Default value is 10.
Collect Execution Detail	Writes a row to the database for every query execution, including aggregate of source access within the query. Useful in application development environment.
Collect Source Detail	Writes a row to the database for every source access in a query. Collect Execution Detail needs to be configured for this parameter to take effect.

Creating a Performance Profiler

This section lists the steps needed to create a performance profiler.

1. Create a table to store the following audit properties:
   - `common/time/timestamp`
   - `query/service/function`
   - `query/performance/evaltime`
   - `common/application/user`
   - `common/application/name`
   - `common/application/server`

   In addition to the above mentioned properties, you will also need to store:
   - information about the audit event exception, if any.
   - audit event severity level, which can be of types I (Information), W (Warning), S (Success), E (Error), F (Failure).
2. Modify the CLASSPATH to include a pointer to the JAR file.
3. Start WebLogic Server.
4. In the Audit page, configure the database tables as required.
5. In the Security Providers page of the WebLogic Administration Console, configure a ALDSP audit provider. See Table 9-11, “Configuration Parameters for Performance Profiling,” on page 9-25 for details.
6. Restart your WebLogic Server.
7. Run the data service application and use the applicable database visualizer to view the results.

Using the Sample Performance Profiler

A ALDSP audit provider sample file `profiler.zip`, is available in the ALDSP root installation directory. The zip file contains the following files:

- `README.txt` lists steps to use the sample audit provider).
- `dsp_profile.sql` files – Contains table definitions.
- `build.xml` – Defines build configurations.
- `DSPProfilerMBean.xml` – MBean definition file for the ALDSP profiling auditor.
- `DSPProfilerImpl.java` – Sample java code that implements the weblogic.security.spi.AuditProvider and weblogic.security.spi.AuditChannel interfaces.

Monitoring the Server Log

Server log files contain information about the time spent to compile and execute a query. The log is in the following location:

`<BeaHome>\user_projects\domains\<domainName>\<serverName>\<server>.log`

For more information about WebLogic Server logs, see Viewing the WebLogic Server Logs.

You can configure the log levels, by application, using the General application configuration page. For more information, see “Configuring the Cache and Log for a Dataspace” on page 4-2. The log levels include:

- **Error.** Runtime exceptions.
- **Notice.** Possible errors that do not affect runtime operation, as well as error level events.
- **Information.** Start/stop events, unsuccessful access attempts, query execute times, and so on, as well as error and notice level events.

Debug logging occurs by default for any server in development mode. Client applications can contribute to the server log through the WebLogic Logger facility. For more information, see Using WebLogic Logging Services.

Query strings are echoed in the server log as a debug-level log message when the log level is set to Information in the ALDSP Console and the WebLogic Administration Console is set to log debug messages to stdout.

Monitoring a WebLogic Domain

You can use the WebLogic Server Administration Console to monitor the health and performance of the domain in which WebLogic is deployed, including resources such as servers, JDBC connection pools, JCA, HTTP, the JTA subsystem, JNDI, and Enterprise Java Beans (EJB).

The domain log is located in the following directory:

```
<BeaHome>\user_projects\domains\<domainName>\<domainName>.log
```

For more information, see "Monitoring a WebLogic Server Domain" in *Configuring and Managing WebLogic Server*.

Using Other Monitoring Tools

You can use performance monitoring tools, such as the OptimizeIt and JProbe profilers, to identify ALDSP application "hot spots" that result in either high CPU utilization or high contention for shared resources.

For more information, see "Tuning WebLogic Server Applications." For a complete list of performance monitoring resources, see "Related Reading" in *WebLogic Server Performance and Tuning*.

CHAPTER 10

Extending Database Support

This chapter explains how to extend the database support of AquaLogic Data Services Platform (ALDSP). Extensions let you provide immediate, dynamic support for unsupported databases and new versions of supported databases. This chapter explains how to extend database support using a feature called the Configurable Relational Provider.

Tip: A sample Configurable Relational Provider file is provided in this chapter. You can copy the sample and use it as a starting point for creating your own customized provider. See "Sample Configurable Relational Provider File" on page 10-8 for the complete listing.

This chapter assumes that you are familiar with XQuery and SQL, especially for more advanced use cases. For suggested background on these subjects with respect to ALDSP, see "Related Reading" on page 10-8.

This chapter includes these topics:

Introduction

The Configurable Relational Provider lets you extend the database support and functionality of ALDSP. The Configurable Relational Provider lets you add or modify database support by configuring an XML file, called a "provider." You can configure the XML provider to extend database support for all but a few advanced cases. See "Using the Configurable Relational Provider" on page 10-13 for details.

This section describes the overall framework for extending ALDSP database support, defines general terms, and lists several use cases for the extension framework.

This section includes these topics:

General Use Cases

This section explains cases where you might consider extending database support using the Configurable Relational Provider.

- Case 1: Adding extended RDMBS support for your database or, if extended support is provided, customizing or extending that support further.

  If you are using ALDSP with base platform database support (see "Relational Providers Included With ALDSP" on page 10-6), it is possible that the database itself can handle more complex constructs, such as expressions and clauses, than are generated by the base platform provider. In this case, users might experience reduced performance. To solve this problem, you can configure an Configurable Relational Provider.

- Case 2: Adding support for a new version of a core database.

  If a new version of a core database is released, ALDSP by default treats it the same as the previously supported version. Obviously, with a new release, there may be features that you want to use, such as improved SQL pushdown. In this case, you can update the database support by extending the relational provider for the core database using the Configurable Relational Provider to add the new pushdown features.

- Case 3: Adding support for a new database that has fewer capabilities than the base platform or is not supported by the core databases.

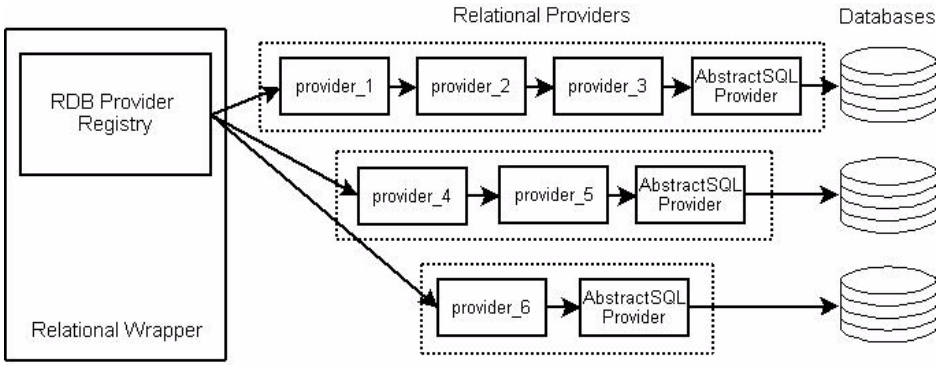
  Note: This use case is uncommon.

  It is possible that you require access to a database that is not supported by ALDSP core database set (see Table 10-1) and that cannot consume SQL generated by the base platform provider. In this case, you can use the Configurable Relational Provider and either disable unsupported features or add new features as desired.

Overview of the Extension Framework Architecture

The Relational Wrapper Extension Framework lets you add or modify relational database support for ALDSP. This framework supports the Configurable Relational Provider, which lets you extend database support by editing a configuration file. Figure 10-1 illustrates the architecture of the Relational Wrapper Extension Framework.

Figure 10-1 Database Extension Framework Architecture



This framework includes a component called a Relational Wrapper that exposes XQuery views of relational sources and executes queries against them. The Relational Wrapper includes the Relational Database Provider Registry, which manages chains of components called relational providers.

Tip: The Configurable Relational Provider, which is discussed in detail in this chapter, is an example of a relational provider that you can easily configure and deploy by editing a file. The Configurable Relational Provider is the primary means by which you can extend database support.

- Defines the SQL and runtime capabilities of a specific database.
- Allows ALDSP to handle different databases and their SQL dialects.
- Returns information about runtime and SQL generation capabilities of the database supported by the provider.
- Can be extended to add support for new databases and customize support for existing ones.

Inside the provider registry, relational providers are organized into chains. These chains delegate to one another and allow method invocations to be intercepted and processed along the way. Each provider either answers a request or delegates the request to its parent provider. A provider's parent is specified by the <parent> element of the provider's deployment descriptor (see "Configurable Relational Provider Format Description and Reference" on page 10-16).

As shown in Figure 10-1, the first chain is assembled from three providers: **provider_1**, **provider_2** and **provider_3**. When the relational wrapper calls this chain, **provider_1** first receives the call and has a choice of either answering it or delegating to its parent provider (**provider_2**). If **provider_1** delegates to **provider_2** then it is the responsibility of **provider_2** to handle the request. In turn, **provider_2** can decide to delegate processing to **provider_3**. This chain architecture increases system flexibility by supporting modular provider definitions and facilitating easy assembly.

Tip: Typically, when you create a provider using the Configurable Relational Provider, you specify a parent provider. The parent provides some features that the child provider can either accept by default or override. The child provider inherits the features of the parent; however, you can also add features to the child provider that are not implemented in the parent. Usually, one of the abstract providers serves as the parent of the first provider in a chain. See "Abstract SQL Providers" on page 10-51.

By default, the Relational Wrapper Extension Framework supports a core set of databases. See "Relational Providers Included With ALDSP" on page 10-6 for a complete list. Extensibility allows for full support of databases that are not in the core set and allows for support of new versions of the core databases. For example, a new version of a core database might provide new pushdown capabilities that are not currently recognized by ALDSP. You can use the extension framework to add the required database support immediately by editing and deploying a Configurable Relational Provider.

Relational Providers Included With ALDSP

Table 10-1 lists the set of standard relational providers that are included with ALDSP. Standard providers are implemented using the Relational Wrapper Extension Framework and are registered by default. You can use these providers as a basis for configuring the Configurable Relational Provider.

Table 10-1 Relational Providers Included With ALDSP

Provider ID	Supported Database Type and Version(s)	Base Database Version (Decimal)
Oracle-8	Oracle >= 8	8
Oracle-9	Oracle >= 9	9
MSSQL-2000	Microsoft SQL Server >= 2000	8 Version 8 is the product version returned by the JDBC drivers for SQL Server 2000.
DB2-8	IBM DB2 >= 8	8
Sybase-12.5.2	Sybase >= 12.5.2	12.52
Pointbase	Pointbase >= 5.1	5.1
Access	Microsoft Access 2003 Microsoft Access support is implemented using the Configurable Relational Provider described in "Using the Configurable Relational Provider" on page 10-13.	4
AbstractSQL, AbstractSQL89, AbstractSQL92	These abstract providers provide base functionality to the Configurable Relational Provider. See "Configurable Relational Provider Format Description and Reference" on page 10-16 for details. See also "Abstract SQL Providers" on page 10-51.	Not applicable. The abstract providers do not match any databases, and therefore do no return a base version.

Tip: The Base Database Version is calculated by the framework. This value specifies the minimum version of a database that a provider can handle. Matching rules are used to determine the value when you pick a provider that best matches your database. For more information on this calculation, see "Database Matching " on page 10-31.

Supported Features

The Configurable Relational Provider supports the following features found in the core relational providers:

- Database matching
- Standard JDBC type mapping
- Join pushdown specification
- Clause pushdown specification
- Function and operator pushdown
- Cast pushdown
- Auto-generation of fields (usually keys)
- Stored procedure configuration
- A subset of runtime properties

Some features defined by the Relational Wrapper Extension Framework are not supported by the Configurable Relational Provider. In such cases, the Configurable Relational Provider delegates the request to its parent provider, which answers it.

The unsupported features include:

- Data type mapping
- Data type based matching when pushing down functions and cast operations
- SQL expression kind matching when pushing down functions and cast operations

Importing Relational Source Metadata

You can import metadata on the data sources needed by your application using the AquaLogic Data Services Platform Metadata Import wizard. This wizard introspects available data sources and identifies data objects that can be rendered as data services and functions. The relational provider registry returns a list of providers that best match the database. You can then pick one of these providers (typically, the best match or one close to the best match) from a drop down menu. The best match appears at the top of the drop down menu. Once created, physical data services become the building-blocks for queries and logical data services. For detailed information on using the Metadata Import wizard, see Creating and Updating Physical Data Services in the *Data Services Developer's Guide*. For information on how matching is performed, see "Rules for Database Matching" on page 10-32.

Related Reading

Refer to the following ALDSP documentation for more information on ALDSP database, XQuery, and SQL support:

- XQuery Engine and SQL in the *XQuery and XQSE Developer's Guide*.
- XQuery-SQL Mapping Reference in the *XQuery and XQSE Developer's Guide*
- "Supported Relational Database Management Systems" in *ALDSP Supported Configurations*

Sample Configurable Relational Provider File

Listing 10-1 shows a sample Configurable Relational Provider file. This sample demonstrates a possible way to configure a custom Microsoft Access provider. You can also find the sample Microsoft Access provider in your ALDSP installation here:

```
ALDSP_HOME\samples\RelationalAdapter\MS-Access
```

Tip: Copy this sample provider to use as a starting point for creating your own customized provider. Reference information in this chapter explains all of the configurable elements of this XML file. To get started, see "Using the Configurable Relational Provider" on page 10-13.

Listing 10-1 Sample Configurable Relational Provider File for a Microsoft Access Database

```
<?xml version="1.0"?>
<aldsp-rdb-extension xmlns="http://www.bea.com/ns/aldsp/rdb/extension">

    <name>MS Access XML Provider</name>
    <vendor>BEA</vendor>
    <implementation-version>1.0</implementation-version>
    <description> MS Access Relational Wrapper Extension </description>


    <rdb-provider>
        <id>MS-Access-2003</id>
        <description>XMLProvider MS Access 2003</description>
        <parent>AbstractSQL</parent>
        <factory
class="com.bea.dsp.wrappers.rdb.providers.custom.XMLCustomizableProviderFactory>
  <custom-rdb-provider
    xmlns="http://www.bea.com/ns/aldsp/rdb/extension/custom"
    xmlns:fn="http://www.w3.org/2004/07/xpath-functions"
    xmlns:fn-bea="http://www.bea.com/xquery/xquery-functions"
    xmlns:op-bea="http://www.bea.com/xquery/xquery-operators"
    xmlns:op="http://www.w3.org/2004/07/xpath-operators"
    xmlns:xdt="http://www.w3.org/2004/07/xpath-datatypes"
    xmlns:xs="http://www.w3.org/2001/XMLSchema">

    <database-kind>
     <match-database>
       <![CDATA[
       (jdbc:getDatabaseProductName() eq "ACCESS") and
       (jdbc:getDatabaseMajorVersion() ge 4)
       ]]>
     </match-database>
     <base-version>4</base-version>
    </database-kind>

    <database-objects>
     <catalog quote="&amp;quot;" separator="." />
     <schema quote="&amp;quot;" separator="." />
     <table quote="&amp;quot;" qualified-name-parts="catalog schema table" />
    </database-objects>
```

```
<joins inner-join="true" outer-join="true">
 <sql92 right-trees="true">
  <inner-join-syntax>
   {0} INNER JOIN {1} ON {2}
  </inner-join-syntax>
 </sql92>
</joins>

<orderby column="true" expression="true" aggregate="true" null-order="low"/>

<groupby column="true" expression="true" constant="true"/>

<subqueries in-from="true" in-where="true" />

<case supported="false" />

<functions>

 <!-- String Functions -->
 <function name="fn:concat" supported="true" infix="true" >&amp;amp;</function>
 <function name="fn:string-length" arity="1">LEN({0})</function>
 <function name="fn:lower-case"
 arity="1">IIF(ISNULL(LCASE({0})),'',LCASE({0}))</function>
 <function name="fn:upper-case" supported="true" >
 IIF(ISNULL(UCASE({0})),'',UCASE({0}))</function>
 <function name="fn:substring" arity="2" >
 IIF(ISNULL(MID({0},{1})),'',MID({0},{1}))</function>
 <function name="fn:substring" arity="3" >
 IIF(ISNULL(MID({0},{1},{2})),'',MID({0},{1},{2}))</function>

 <function name="fn-bea:left" >LEFT({0},{1})</function>
 <function name="fn-bea:right" >RIGHT({0},{1})</function>
 <function name="fn-bea:repeat" supported="false" />
 <function name="fn-bea:trim" arity="1" >TRIM({0})</function>
 <function name="fn-bea:trim-left" arity="1" >LTRIM({0})</function>
 <function name="fn-bea:trim-right" >RTRIM({0})</function>
 <function name="fn-bea:sql-like" arity="2" >({0} LIKE {1})</function>
 <function name="fn-bea:sql-like" arity="3" supported="false" />
 <function name="fn:starts-with" supported="false" />
```

```
 <function name="fn:ends-with" supported="false" />
 <function name="fn:contains" supported="false" />
 <function name="op-bea:string-not-equal" arity="2" >({0} &lt;&gt;
 {1})</function>

 <!-- Numeric Functions -->
 <function name="fn:abs" supported="true" arity="1" >ABS({0})</function>
 <function name="fn:ceiling" supported="false" />
 <function name="fn:floor" supported="false" />
 <function name="fn:round" >ROUND ({0})</function>

 <!-- Aggregate Functions -->
 <function name="fn:count" supported="true" arity="1" >COUNT({0})</function>
 <function name="fn:avg" >AVG({0})</function>
 <function name="fn:min" arity="1" >MIN({0})</function>
 <function name="fn:max" supported="true" arity="1" >MAX({0})</function>
 <function name="fn:sum" arity="1" >
 IIF(ISNULL(SUM({0})),0,SUM({0}))</function>

 <!-- DateTime Functions -->
 <function name="fn:day-from-date" arity="1" >DAY({0})</function>
 <function name="fn:month-from-date" >MONTH({0})</function>
 <function name="fn:year-from-date" >YEAR({0})</function>
 <function name="fn:day-from-dateTime" arity="1" >DAY({0})</function>
 <function name="fn:month-from-dateTime" >MONTH({0})</function>
 <function name="fn:year-from-dateTime" >YEAR({0})</function>
 <function name="fn:hours-from-dateTime" >HOUR({0})</function>
 <function name="fn:minutes-from-dateTime" arity="1" >MINUTE({0})</function>
 <function name="fn:seconds-from-dateTime" >SECOND({0})</function>
 <function name="fn:current-date" supported="false"/>
 <function name="fn:current-time" supported="false"/>
 <function name="fn:current-dateTime" supported="false"/>



</functions>

<casts>
 <cast from="xs:string" from-subtypes="true" to="xs:int">
    CINT({0})
 </cast>
```

```
<cast from="xs:double" from-subtypes="true" to="xs:int">
   CINT({0})
</cast>
<cast from="xs:float" from-subtypes="true" to="xs:int">
   CINT({0})
</cast>
<cast from="xs:decimal" from-subtypes="true" to="xs:int">
   CINT({0})
</cast>

<cast from="xs:string" from-subtypes="true" to="xs:double">
   CDBL({0})
</cast>
<cast from="xs:decimal" from-subtypes="true" to="xs:double">
   CDBL({0})
</cast>
<cast from="xs:string" from-subtypes="true" to="xs:float">
   CDBL({0})
</cast>
<cast from="xs:decimal" from-subtypes="true" to="xs:float">
   CDBL({0})
</cast>

<cast from="xs:string" from-subtypes="true" to="xs:dateTime">
   CDATE({0})
</cast>

<cast from="xs:float" from-subtypes="true" to="xs:string" >
  CSTR({0})
</cast>
<cast from="xs:double" from-subtypes="true" to="xs:string" >
  CSTR({0})
</cast>
<cast from="xs:decimal" from-subtypes="true" to="xs:string" >
  CSTR({0})
</cast>
<cast from="xs:boolean" from-subtypes="true" to="xs:string" >
  CSTR({0})
</cast>
<cast from="xs:dateTime" from-subtypes="false" to="xs:string" >
  CSTR({0})
```

```
     </cast>
    </casts>

    <limit>
     <select-top />
    </limit>

    <insert>
     <auto-column-generator kind="sql-post" >
      select @@identity
     </auto-column-generator>
    </insert>

    <properties
      supports-multiple-active-queries-per-connection="false"
      supports-cancel-query="false"
      supports-query-timeout="false" />

   </custom-rdb-provider>
       </factory>
    </rdb-provider>

</aldsp-rdb-extension>
```

Using the Configurable Relational Provider

This section explains how to use the Configurable Relational Provider. The Configurable Relational Provider lets you configure a new relational provider by editing an XML configuration file.

Tip: Be sure to review the section "Introduction" on page 10-2 before continuing.

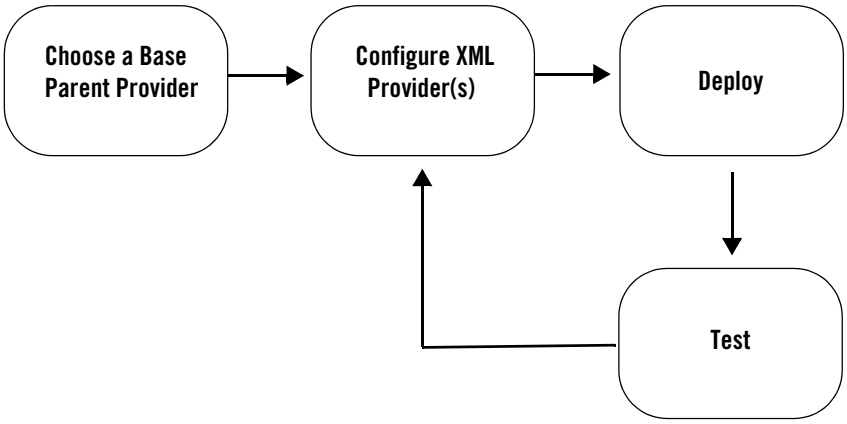
This section includes these topics:

- Summary of Basic Configuration Steps
- Deploying the Relational Provider

Summary of Basic Configuration Steps

This section lists the basic steps required to develop and deploy an Configurable Relational Provider. The basic process of creating a new provider is also shown in Figure 10-2.

Figure 10-2 Custom Provider Development Process



1. Choose a base parent provider, such as one of the Abstract providers discussed in "Abstract SQL Providers" on page 10-51. The base provider represents the first provider in a provider chain. Subsequent providers in the chain can extend or override features of a parent provider. See "Overview of the Extension Framework Architecture" on page 10-4 for information about provider chains.
2. Configure one or more Configurable Relational Providers. Configurable Relational Providers are configured in an XML file in which you specify all of the properties of the Configurable Relational Provider(s). See "Sample Configurable Relational Provider File" on page 10-8. The sample is a good starting point for developing your own customized provider.
3. Deploy the provider. A command line script is provided to deploy your customized provider. See "Deploying the Relational Provider" on page 10-15.
4. Test the provider.

Deploying the Relational Provider

A command-line deployment tool, described in this section, is provided with ALDSP. Use this tool to add and remove relational providers. To use this deployment tool, your provider's deployment descriptor must be packaged in a JAR file. See "Deploying the Relational Provider" on page 10-15.

Note: When ALDSP loads an extension, the deployment descriptor is read and validated. If a provider section of the description is determined to be invalid, it is ignored.

Adding a Provider

The command syntax for adding a provider is:

```
<ALDSP_HOME>/bin/update-providers.[cmd/sh] -add <provider.jar>
```

The fully-qualified path to the provider relational wrapper extension JAR file is required. When a new provider is added, it is copied into the `<ALDSP_HOME>/providers` directory.

Note: Adding or removing a provider requires that you restart the IDE or the server.

Removing a Provider

The command syntax for removing a provider is:

```
<ALDSP_HOME>/bin/update-providers.[cmd/sh] -remove <provider.jar>
```

Specify the filename of the provider JAR file located in the `<ALDSP_HOME>/providers` directory. When an existing provider is removed, it is deleted from the `<ALDSP_HOME>/providers` directory.

Note: Adding or removing a provider requires that you restart the IDE or the server.

Configurable Relational Provider Format Description and Reference

This section describes the format, elements, and configurable properties of an Configurable Relational Provider.

Note: A complete provider example is listed in "Sample Configurable Relational Provider File" on page 10-8.

This section includes:

- Overview of Primary XML Elements – This section provides an overview of the top-level elements of the Configurable Relational Provider.
- Overview of the <custom-rdb-provider> Element – This section provides an overview of the <custom-rdb-provider> element. This element contains all of the sub-elements and properties that define a Configurable Relational Provider.
- Configurable Relational Provider Reference – This section describes all of the elements of the <custom-rdb-provider> element.

Overview of Primary XML Elements

This section describes each of the primary elements in an Configurable Relational Provider file. This file is a deployment descriptor that is used to specify the properties of the relational provider extension.

Tip: The file must be packaged and deployed in a JAR file. The JAR must only contain one deployment descriptor; however, the descriptor can define and configure one or more providers. See "Deploying the Relational Provider" on page 10-15.

The following list describes the primary elements of a relational provider deployment descriptor.

Note: You must name the deployment descriptor file `aldsp-rdb-extension.xml`.

- **`<name>`** – The name of the provider.
- **`<vendor>`** – (Optional) The name of the vendor of the provider.
- **`<implementation-version>`** – (Optional) A version number for the provider.
- **`<description>`** – (Optional) A brief description of the extension.

- **<id>** – The provider ID. This ID is used to register the provider in the provider registry.
- **<description>** – (Optional) A brief description of the provider.
- **<parent>** – (Optional) The <id> element of a parent provider.

Tip: In the sample file in "Sample Configurable Relational Provider File" on page 10-8, the class specified by the <parent> element is AbstractSQL. See "Abstract SQL Providers" on page 10-51 for detailed information on this abstract provider parent class.

- **<modifier>** – (Optional) Either `abstract` or `final`. If set to `abstract`, the provider cannot be referred to by any data service; however, an abstract provider can be extended (be the parent of another provider). If set to `final`, the provider cannot be extended by any other providers.
- **<factory>** – (Optional) This element specifies a factory class that instantiates the provider. The Configurable Relational Provider uses the default factory class, ,XMLCustomizableProviderFactory.

Tip: In the sample file in "Sample Configurable Relational Provider File" on page 10-8, the <factory> element explicitly specifies the default factory class, XMLCustomizableProviderFactory.

- **<custom-rdb-provider>** – A sub-element that specifies the namespace of the Configurable Relational Provider and its full configuration. The default namespace is: `http://www.bea.com/ns/aldsp/rdb/extension/custom`.

Tip: For details on configuring the <custom-rdb-provider> element, see "Sample Configurable Relational Provider File" on page 10-8 and "Configurable Relational Provider Format Description and Reference" on page 10-16.

Note: When ALDSP loads an extension, the deployment descriptor is read and validated. If a provider section of the description is determined to be invalid, it is ignored.

Overview of the <custom-rdb-provider> Element

Listing 10-2 shows the basic configuration of the `<custom-rdb-provider>` element in an Configurable Relational Provider. This configuration is based on a schema file that is provided with ALDSP.

Each of the properties are described in greater detail in "Configurable Relational Provider Reference" on page 10-20. For a complete example, see "Sample Configurable Relational Provider File" on page 10-8.

Listing 10-2 Overview of the <custom-rdb-provider> Element

```
<custom-rdb-provider xmlns="http://www.bea.com/ns/aldsp/rdb/extension/custom">

    <database-kind>
        <match-database>
            XQuery expression that uses a predefined external function to
            Access JDBC metadata. Result type: boolean
        </match-database>
        <base-version>
            Base database version supported by this provider (decimal)
        </base-version>
        <matched-version>
            XQuery expression returning matched version. Result type: decimal
        </matched-version>
    </database-kind>

    <database-objects>
        <catalog   quote?="string" separator?="string"/>
        <schema    quote?="string" separator?="string" />
        <table     quote?="string" separator?="string"
                                   qualified-name-parts="string*"/>
        <column    quote?="string" />
        <procedure quote?="string" qualified-name-parts="string*"/>
    </database-objects>

    <joins inner-join="boolean" outer-join="boolean">
        <sql92 right-trees="boolean(:=true)" />  or
        <sql89 outer-join-kind?="columnModifier|tableModifier"
               outer-join-modifier?="string" />
```

```
</joins>

<orderby column?="boolean" expression?="boolean" aggregate?="boolean"
         null-order?="low|high|first|last|undefined"
         style?="ordering-expression|ordering-expression-with-projection|
            position-in-project-list" />

<groupby column?="boolean" constant?="boolean" expression?="boolean" />

<subqueries in-from?="boolean" in-where?="boolean" />

<case supported?="boolean(:=true)" />

<functions default-syntax-for-empty-input="lax|strict|strict-coalesce">
    <function name="QName" arity?="integer" supported?="boolean(:=true)"
       infix?="boolean(:=false)">
       SQL expression which uses {0},{1},…{n} for input expressions
       (string)
    </function>
</functions>

<casts>
    <cast from="QName" from-subtypes?="boolean(:=false)" to="QName"
          supported?="boolean(:=true)">
        SQL expression which uses {0} for input expression
    </cast>
</casts>

<limit supported?="boolean(:=true)">
    <top parameter="true|false" composable="true|false"/> or
    <rownum kind="project_first|filter_first">
        ROWNUM
    </rownum>
</limit>

<insert>
    <key-gen kind?="jdbc|sql-pre|sql-post">
         SQL statement
    </key-gen>
</insert>
```

```
    <properties
         supports-query-timeout = "boolean"
         supports-cancel-query  = "boolean"
         supports-multiple-active-queries-per-connection = "boolean"
    />

</custom-rdb-provider>
```

Configurable Relational Provider Reference

Table 10-2 describes each of the sub-elements and properties of the <custom-rdb-provider> element of an XML Customization Provider configuration file.

For a summary of the file format, see "Overview of the <custom-rdb-provider> Element" on page 10-18. For a complete example, see "Sample Configurable Relational Provider File" on page 10-8.

Tip: Most of the settings listed in Table 10-2 are optional. Any settings that are specified in the configuration file override default settings provided by the parent provider. The parent provider is specified with the <parent> element of the descriptor. If no setting is provided for an attribute, then the request is delegated to the parent provider. See "Overview of the Extension Framework Architecture" on page 10-4 for a description of the way in which providers delegate to parent providers in a "chains."

Table 10-2 Configuration Elements and Attributes Description

Element(s) and • Sub-element(s)	Attribute(s)	Description of Element or Attribute
<match-database> <matched-version>		These elements contain XQuery expressions that can access JDBC database metadata through predefined external functions. See "Database Matching " on page 10-31. There are no default values for these elements. Default values are inherited from the parent provider.
<database-objects>		Sub-elements of this element specify various properties of database object identifiers in the generated SQL.

Table 10-2 Configuration Elements and Attributes Description (Continued)

Element(s) and • Sub-element(s)	Attribute(s)	Description of Element or Attribute
<database-objects> • <catalog> • <schema> • <table> • <column> • <procedure>	quote	The quote attribute specifies the identifier quote for the corresponding database object. Example: <catalog quote=”’” /> To specify open and close quotes, specify first the open quote, then the close quote. Example: <table quote=”[]” /> The general rule is: if the number of characters in the specified quote string is even – then it is assumed that open and close quotes are different. The first half of the specified string is the open quote; the second half is the close quote. If the number of characters in the specified string is odd then it is assumed that the open and close quotes are the same and equal to the whole string.
<database-objects> • <catalog> • <schema> • <table> • <procedure>	separator	The separator attribute specifies the separator character between object identifiers in the fully qualified object name. Example: <schema separator=”.”/> If this attribute is not specified, the parent provider’s value is used by default.
<table> <procedure>	qualified-name-parts	The qualified-name-parts attribute specifies a list of object kinds that specify how a fully qualified name is constructed for this database object. **Note:** Object kinds in the list must be separated by a space character. Example: <table qualified-name-parts=”catalog schema table” /> Example: <procedure qualified-name-parts=”schema procedure”/> If this attribute is not specified, the parent provider’s value is used by default.

Table 10-2 Configuration Elements and Attributes Description (Continued)

Element(s) and • Sub-element(s)	Attribute(s)	Description of Element or Attribute
<joins>	inner-join outer-join	These attributes are booleans that specify whether the database supports inner and outer joins respectively. The exact join syntax is defined by the sql92 and sql89 child elements of the joins element. Example: <joins inner-join="true" outer-join="true"> If these attributes are not specified, the parent provider's values are used by default.
<joins> • <sql92>		The sql92 sub-element specifies that the database uses SQL-92 syntax for joins. For example: SELECT ... FROM a INNER JOIN | LEFT OUTER JOIN b ON ...
<joins> • <sql92>	right-trees	This attribute is a boolean that determines whether parenthesis can be used to control the order of joins in the join clause. Default: true
<joins> • <sql92>	inner-join-syntax	(Optional) Defines the syntax for an inner join. {0} is used for the left branch source, {1} for the right branch source, and {2} for a join condition expression. Example: {0} JOIN {1} ON {2}
<joins> • <sql92>	outer-join-syntax	(Optional) Defines the syntax for a left outer join. {0} is used for the left branch source, {1} for the right branch source, and {2} for the join condition expression. Example: {0} LEFT OUTER JOIN {1} ON {2}
<joins> • <sql89>		The sql89 sub-element specifies that the database uses SQL-89 syntax for joins. For example: SELECT ... FROM a,b WHERE ...
<joins> • <sql89>	inner-join-syntax	(Optional) Defines the syntax for a left inner join. {0} is used for the left branch source, {1} for the right branch source. Default: {0}, {1}

Table 10-2 Configuration Elements and Attributes Description (Continued)

Element(s) and • Sub-element(s)	Attribute(s)	Description of Element or Attribute
<joins> • <sql89>	outer-join-syntax	(Optional) Defines the syntax for a left outer join. {0} is used for the left branch source, {1} for the right branch source. Example: {0}, OUTER {1} Default: empty (left outer join is not supported)
<joins> • <sql89>	outer-join-right-branch-column-modifier	(Optional) Specifies the transformation to be applied to the columns on the right side of a left outer join. {0} is used to specify the right-side column. Example: {0}(+) Default: empty (no transformation is required)
<orderby>	column expression	This boolean attribute specifies whether the database supports orderby column and other expressions. If these attributes are not specified, the parent provider's values are used by default.
<orderby>	aggregate	This boolean attribute specifies whether the database supports orderby aggregate.
<orderby>	null-order	This attribute specifies one of the following values: • low – NULL values are sorted low. • high – NULL values are sorted high. • first – NULL values are sorted at the start regardless of sort order. • last – NULL values are sorted at the end regardless of sort order. • undefined – NULL values are sorted by ALDSP ("order by" is not pushed to the database in this case). If this attribute is not specified, the parent provider's values are used by default.

Table 10-2 Configuration Elements and Attributes Description (Continued)

Element(s) and • Sub-element(s)	Attribute(s)	Description of Element or Attribute
<orderby>	style	Style of the orderby expressions that will be generated: • position-in-project-list – Generates ORDER BY n, where 'n' is a position of the ordering expression in the SELECT clause. The ordering expression is automatically added to the SELECT clause if necessary. • ordering-expression-with-projection – Generates ORDER BY <expr> where <expr> is automatically added to the SELECT clause if necessary. • ordering-expression – Generates ORDER BY <expr> where <expr> is not automatically added to the SELECT clause. There is no default value for this attribute. The parent provider's value is used if not specified.
<groupby>	column constant expression	These boolean attributes specify whether the group by clause can operate on columns, constants, and expression. If these attributes are not specified, the parent provider's values are used by default.
<subqueries>	in-from in-where	These boolean attributes specify whether subqueries are supported in FROM and WHERE clauses. ALDSP generates only a subquery in the WHERE clause only when translating a semi-join. Example: "WHERE EXITS(...))") If these attributes are not specified, the parent provider's values are used by default.
<case>	supported	This boolean attribute specifies whether the CASE expression is supported. Default: true
<functions>		This element defines SQL syntaxes for functions.

Table 10-2 Configuration Elements and Attributes Description (Continued)

Element(s) and • Sub-element(s)	Attribute(s)	Description of Element or Attribute
<functions>	default-syntax-for-empty-input	An enumeration of strings that define which default syntax to use in the presence of NULL input. NULL (an empty sequence in XQuery) input is usually handled differently by SQL and XQuery functions. In SQL, NULL is usually propagate to the output of a function. For example: f(NULL)=NULL). In XQuery, however, NULL is usually replaced with a value. For string functions, such as f(()) = "", sum(())=0, and so on. This setting specifies how to deal with such situations when choosing default SQL syntax for a function. This attribute must specify one of the following values: • strict – Follow XQuery semantics. Do not push down if the input can be empty. • strict–coalesce – (Default) Follow XQuery semantics. Push down with the help of the COALESCE function in SQL. Only use this value if the database supports the COALESCE function. • lax – Do not follow XQuery semantics. Generate SQL without the COALESCE function, such that f(NULL) -> NULL. See "Default SQL Syntax for Functions" on page 10-36. Default: strict-coalesce

Table 10-2 Configuration Elements and Attributes Description (Continued)

Element(s) and • Sub-element(s)	Attribute(s)	Description of Element or Attribute
<functions> • <function>		This sub-element defines the translation of an XQuery function(operator) into SQL. The contents of this sub-element is a SQL expression that must be generated for the named function. Parameters are specified as {0}, {1}, … {n}. A variable number of parameters is supported. See "Specifying SQL Syntax for Functions" on page 10-34 for more information on the format. This element is not required if the supported attribute is set to false. The contents of this element can be empty. In this case, the default syntax for this function is used for SQL generation. A list of default syntaxes is provided in "Default SQL Syntax for Functions" on page 10-36. For examples, see "Sample Configurable Relational Provider File" on page 10-8.
<functions> • <function>	name	(Required) Specifies the QName of a function. See "Function and Type Name Resolution Process" on page 10-51.
<functions> • <function>	arity	Specifies the arity of the named function. Can be omitted if function name is non-ambiguous.
<functions> • <function>	supported	(Boolean) specifies whether the function pushdown is supported or not. Disables the pushdown of a function defined by the parent provider. Default: true
<functions> • <function>	infix	(Boolean) Specifies whether or not to use infix formatting style for this function. A SQL expression in the sub-element contents specifies the only infix operation in this case. Parameters are processed automatically.
<casts>		This element defines cast operations for push down.

Table 10-2 Configuration Elements and Attributes Description (Continued)

Element(s) and • Sub-element(s)	Attribute(s)	Description of Element or Attribute
<casts> • <cast>		Defines translation for cast operations to SQL for a particular combination of types. The content of this element is the SQL expression that must be generated for this cast operation. The parameter is specified as {0}. This element is not required if the supported attribute is false. For examples, see "Sample Configurable Relational Provider File" on page 10-8.
<casts> • <cast>	to from	These attributes specify the QNames of input and target XQuery types. If only a local name is specified, ALDSP searches for the type in well-known namespaces. For examples, see "Sample Configurable Relational Provider File" on page 10-8. See also "Standard and ALDSP Namespaces for Functions and Types" on page 10-50.
<casts> • <cast>	from-subtypes	(Boolean) Specifies whether the matching input type must also match its subtypes (according to XQuery type hierarchy). Default: false. For examples, see "Sample Configurable Relational Provider File" on page 10-8.
<casts> • <cast>	supported	(Boolean) Specifies whether this cast operation is supported. Intended usage is to disable cast pushdown of the parent provider. Default: true
<limit>		This element defines the pushdown of fn:subsequence(). This element must have one child element specified. To disable pushdown of this function, set supported to false.
<limit>	supported	(Boolean) Specifies whether the database supports fn:subsequence() pushdown. Default: true

Table 10-2 Configuration Elements and Attributes Description (Continued)

Element(s) and • Sub-element(s)	Attribute(s)	Description of Element or Attribute
<limit> • <select-top>		Specifies that fn:subsequence() must be pushed down using the TOP modifier of the SELECT clause. For example: SELECT TOP n FROM ... The content of the select-top element defines SQL syntax for the select clause modifier. {0} is bound to the length expression. Default content value: TOP {0}
<limit> • <select-top>	parameter	(Boolean) Specifies whether the TOP value can be a parameter. For example, whether SELECT TOP ? FROM ... is supported by the database. Default: false
<limit> • <select-top>	composable	If set to true, specifies whether to stop SQL generation after processing fn:subsequence(). If set to false, continues by creating a subquery for a SELECT TOP ... statement. Default: false
<limit> • <row-number-function>		Specifies that the fn:subsequence() is a pushdown using a ROWNUM-like function. The content of this element defines the SQL syntax for ROWNUM-like functions supported by the database. The content portion is optional. Default content: ROW_NUMBER() Over(...)
<limit> • <row-number-function>	explicit-order-by	(Boolean) Determines whether ORDER BY ordering expressions will be passed as arguments to the ROW_NUMBER function.
<limit> • <row-number-function>	split-range-filter	(Boolean) Determines whether the range test should be split between subqueries. (Oracle ROWNUM pattern) Default: false

Table 10-2 Configuration Elements and Attributes Description (Continued)

Element(s) and • Sub-element(s)	Attribute(s)	Description of Element or Attribute
<limit> • <limit-clause>		Specifies that fn:subsequence() should be translated into SQL as a LIMIT-like clause added at the end of a SQL query. Content of the <limit-clause> element defines SQL syntax for this clause, where {0} and {1} placeholder bindings depend on the @style attribute (see below). Content value is optional. Default content value: LIMIT {0} OFFSET {1}
<limit> • <limit-clause>	kind	Defines kind of the accepted subsequence() function: • Range - default - both start and length expression are used. In this case limit clause syntax has {0} parameter bound to the start expression and {1} to the length expression • Top - only top-like subsequence() is accepted for pushdown. start expression has to be constant 1. In this case limit clause syntax has only {0} parameter which is bound to the length expression Default value: range
<limit> • <limit-clause>	parameter	(Boolean) Specifies whether SQL parameters can be used in limit clause (as start and/or length expressions) Default value: true
<limit> • <limit-clause>	composable	(Boolean) Specifies whether SQL generation should stop after processing fn:subsequence() (when set to false), or can continue by creating subquery for SELECT … LIMIT statement (when set to true). Default: false
<limit> <limit-clause>	start-base	Integer. 0 or 1. Defines whether start expression is 0 or 1 - based. Only applicable when @style = range Default: 0

Table 10-2 Configuration Elements and Attributes Description (Continued)

Element(s) and • Sub-element(s)	Attribute(s)	Description of Element or Attribute
<insert> • <auto-column-generator>		Defines a strategy to access auto-generated columns when inserting data into the database. Strategy kind is defined by the kind attribute. The content of this element is a SQL expression for certain kinds and empty for others. Example: <insert> <auto-column-generator kind="sql-post"> SELECT LAST_INSERT_ID() </auto-column-generator></insert>
<insert> • <auto-column-generator>	kind	A string constant (enumeration) that defines the key generation strategy. This attribute must specify one of the following values: • jdbc – (Default) Defines the key generation strategy through the JDBC API. Content of the key-gen element must be empty in this case. • sql-pre – Run a specified SQL statement to get the auto-generated key before issuing an INSERT statement. For example, use this attribute to get a key from a sequence-like database object. In this case, the content of the key-gen element is the SQL statement that can use {0} as a placeholder for the sequence object name (specified by the dataservice's annotation). • sql-post – Run a specified SQL statement to get the auto-generated key after an INSERT statement. The content of the key-gen element is the SQL statement that must be executed.
properties		The attributes of this element contain various SQL generation and execution properties. These properties do not have default values; the parent's value is used if a property is not set.
<properties>	multiple-active-queries-per-connection-supported	(Boolean) Specifies whether the database supports multiple active statements open on the same connection.

Table 10-2 Configuration Elements and Attributes Description (Continued)

Element(s) and • Sub-element(s)	Attribute(s)	Description of Element or Attribute
<properties>	cancel-query-supported	(Boolean) Specifies whether the jdbc.sql.Statement.cancel() method is supported by the database and driver.
<properties>	query-timeout-supported	(Boolean) Specifies whether the jdbc.sql.Statement.setQueryTimeout() method is supported by the database and driver.

Database Matching

This section describes how ALDSP determines the best database match for a given provider. Database matching logic is specified as an XQuery expression that can access JDBC database metadata through predefined XQuery external functions.

Matching expressions are specified in the Configurable Relational Provider elements (Table 10-2, “Configuration Elements and Attributes Description,” on page 10-20) and are evaluated by the ALDSP XQuery engine. Expressions can use standard XQuery functions supported by the ALDSP XQuery engine as well as additional functions defined by the Configurable Relational Provider. Database matching XQuery expressions return an xs:boolean value.

Another use of matching XQuery expressions is to compute the matched database version (in this case the result must be xs:decimal).

This section includes these topics:

- Rules for Database Matching
- JDBC Metadata Methods to XQuery Functions Mapping
- Additional External XQuery Functions

Rules for Database Matching

The framework employs matching rules to determine if a given provider is compatible with a database. During the metadata import process (see "Importing Relational Source Metadata" on page 10-8) the relational provider registry determines which providers support the database being imported. For successful matches, the base version offset is also obtained. The base version offset is calculated as:

Base version offset (decimal) = (matched db version – base db version returned by the provider)

Base version decimals for the standard providers are listed in Table 10-1 in the section "Relational Providers Included With ALDSP" on page 10-6.

The Datasource Import Wizard uses the base version offset to display providers when there are multiple matches. The wizard's drop down menu contains providers with the minimum base version offset (that is, the closest version to the database). The best match appears at the top of the drop down menu. For information on the Datasource Import Wizard, see Creating and Updating Physical Data Services in the *Data Services Developer's Guide*.

For example, consider the standard DB2 relational provider. This provider matches all DB2 versions starting from 8. Its base version is 8. Assume that a new DB2 provider is created with the Configurable Relational Provider that matches DB2 9 with base version 9. During metadata import of a table from the DB2 9 instance, both providers will match the database. However, for the first provider, the base version offset is 1, but the second one is be 0. Therefore, the second provider will be preferred over the first one.

JDBC Metadata Methods to XQuery Functions Mapping

This section describes the mapping of a java.sql.DatabaseMetaData instance to a set of XQuery functions that can be used by a database matching expression.

Mapped interface: java.sql.DatabaseMetaData

Function namespace:

- `prefix = jdbc`
- `uri = http://www.bea.com/ns/aldsp/extensions/rdb/providers/custom/jdbc`

Requirements for mapped methods:

- No parameters
- Return type of: boolean, string, or int

Table 10-3 lists the java.sql.DatabaseMetaData methods that satisfy these requirements and their corresponding JDBC methods and XQuery functions.

Table 10-3 Java Method to XQuery Function Mapping

Java Method	XQuery Function
int getDatabaseMajorVersion()	jdbc:getDatabaseMajorVersion() as xs:int?
int getDatabaseMinorVersion()	jdbc:getDatabaseMinorVersion() as xs:int?
String getDatabaseProductName()	jdbc:getDatabaseProductName() as xs:string?
String getDatabaseProductVersion()	jdbc:getDatabaseProductVersion() as xs:string?
int getDriverMajorVersion()	jdbc:getDriverMajorVersion() as xs:int?
int getDriverMinorVersion()	jdbc:getDriverMinorVersion() as xs:int?
String getDriverName()	jdbc:getDriverName() as xs:string?
String getDriverVersion()	jdbc:getDriverVersion() as xs:string?
String getURL()	jdbc:getURL()as xs:string?

Exception handling:

- SQLException, RuntimeException – Rethrows the exception.
- LinkageError – Returns an empty sequence. This exception occurs if the driver is compiled against older version of JDBC API.

Additional External XQuery Functions

This section describes additional functions that are available in the database matching expression, but are not directly mapped from the jdbc.sql.DatabaseMetaData interface.

Function namespace:

- `prefix = cxp`
- `uri = http://www.bea.com/ns/aldsp/extensions/rdb/providers/custom/`

Table 10-4 lists and describes the function signatures.

Table 10-4 Function Signatures

Function signature	Description
cxp:getDatabaseVersion() as xs:decimal	Returns the database version as xs:decimal. The version is computed based on java.sql.DatabaseMetaData as follows: 1. Try to detect the version from the string returned by the getDatabaseProductVersion() method. Search for a format: n1.n2.n3. n1, n2, n3 must be non-negative integers and n3 is optional. The resulting decimal version is n1+max(n2,99)*0.01+max(n3,999)*0.00001 2. If Step 1 fails and if getDatabaseMajorVersion(), getDatabaseMinorVersion() are implemented by the driver, then the result is: major + max(minor,99)*0.01
cxp:getDriverVersion() as xs:decimal	Same as approach 1, but uses the following functions from jdbc.sql.DatabaseMetaData: getDriverVersion(), getDriverMajorVersion(), getDriverMinorVersion().

Specifying SQL Syntax for Functions

This section discusses the SQL syntax for functions specified in the Configurable Relational Provider deployment descriptor. See also "Configurable Relational Provider Reference" on page 10-20 and the example descriptor in "Sample Configurable Relational Provider File" on page 10-8.

This section includes these topics:

Syntax Overview

Function SQL syntax is specified as a string with placeholders for each parameter. The syntax defines a SQL fragment to be generated by the relational wrapper when translating the corresponding XQuery function into SQL. It is specified as the content of the `<function>` element.

Example:

```
<function name=”fn:lower”>LOWER({0})</functions>
```

Parameter placeholders start with 0. There can be more than one placeholder with the same index which means that the argument must be replicated in the generated SQL.

Example:

```
<function name=”fn:substring” arity=”2”>SUBSTR({0}, {1}, LENGTH({0})-{1}+1)
</function>
```

Functions with a variable number of arguments can be specified in two different ways:

- By setting the `infix` attribute and specifying only a delimiter as the function syntax
- By using a variable length placeholder: **`{...}`**

These methods are described in the next two sections.

Setting the infix Attribute

The infix attribute of the function element is set as follows:

```
<function name=”fn:concat” infix=”true”>||</function>
```

The generated SQL for this example is:

```
arg1 || arg2 || arg3 || ... || argN
```

Using a Variable Length Placeholder

During SQL generation the variable length placeholder `{...}` is replaced with the remaining arguments separated by commas.

```
<function name=”fn:concat”>CONCAT({...})</function>
```

The generated SQL is:

```
CONCAT(arg1,arg2,arg3,….,argN)
```

If another delimiter is required, it must be specified inside the variable length placeholder as follows:

```
{...DELIMITER}
```

For example:

```
<function name=”fn:concat”>COALESCE({... || }, ””)</function>
```

The generated SQL is:

```
COALESCE(arg1 || arg2 || arg3 || … || argN, "")
```

Note: In this case the delimiter is "`||` ".

Default SQL Syntax for Functions

The default syntax for a function is used when the function is specified in the `<functions>` section of the Configurable Relational Provider configuration file (Table 10-2), but its syntax is not provided by the user (the `<function>` element content is empty). For some functions in this case, the relational provider chooses default syntax based on the `default-syntax-for-empty-input` attribute. See "Configurable Relational Provider Reference" on page 10-20 for information on the `default-syntax-for-empty-input` attribute.

This section lists the default syntaxes used for the three possible values of the `default-syntax-for-empty-input` attribute.

Attribute	Described In
strict	Table 10-5
strict-coalesce	Table 10-6
lax	Table 10-7

Functions for which the default SQL syntax depends on the default-syntax-for-empty-input attribute are denoted with an asterisk (*) in Table 10-5, Table 10-6, and Table 10-7.

These functions are:

- fn:concat
- fn:substring with 2 parameters
- fn:substring with 3 parameters
- fn:string-length
- fn:lower-case
- fn:upper-case
- fn:sum

If default syntax is not defined for a function, then you must specify the syntax of the function when you use it. Otherwise, it is an error.

Table 10-5 default-syntax-for-empty-input = strict-coalesce

XQuery function	Default SQL syntax	Pushdown requirements
op:numeric-add	{0} + {1}	
op:numeric-multiply	{0} * {1}	
op:numeric-divide	{0} / {1}	
op:numeric-mod	MOD({0}, {1})	
fn:abs	ABS({0})	
fn:ceiling	CEILING({0})	
fn:floor	FLOOR({0})	
fn:round	FLOOR({0} + 0.5)	
fn-bea:sql-round	ROUND({0})	
* Functions for which the default SQL syntax depends on the `default-syntax-for-empty-input` setting.		

Table 10-5 default-syntax-for-empty-input = strict-coalesce (Continued)

XQuery function	Default SQL syntax	Pushdown requirements
* fn:concat	COALESCE({0} || {1} || … || {n}, '') COALESCE is not used if at compile-time it is determined that input can never be empty (NULL).	Arguments are not of type CLOB or LONG VARCHAR.
* fn:substring ($str, $pos)	if $pos is a subtype of xs:integer COALESCE(SUBSTRING({0} FROM {1}), '') else COALESCE(SUBSTRING({0} FROM CAST({1}+0.5 AS INTEGER)), '') COALESCE is not used if at compile-time it is determined that input can never be empty (NULL).	First argument is not of type CLOB or LONG VARCHAR.
* fn:substring($str, $pos, $len)	if $pos and $len are subtypes of xs:integer COALESCE(SUBSTRING({0} FROM {1} FOR {2}), '') else COALESCE(SUBSTRING({0} FROM CAST({1}+0.5 AS INTEGER) FOR CAST({2}+0.5 AS INTEGER)), '') COALESCE is not used if at compile-time it is determined that input can never be empty (NULL).	First argument is not of type CLOB or LONG VARCHAR.
* fn:string-length	COALESCE(CHAR_LENGTH({0}), 0) COALESCE is not used if at compile-time it is determined that input can never be empty (NULL).	Argument is not of type CLOB or LONG VARCHAR.
* Functions for which the default SQL syntax depends on the `default-syntax-for-empty-input` setting.		

Table 10-5 default-syntax-for-empty-input = strict-coalesce (Continued)

XQuery function	Default SQL syntax	Pushdown requirements
* fn:lower-case	COALESCE(LOWER({0}), '') COALESCE is not used if at compile-time it is determined that input can never be empty (NULL).	Argument is not of type CLOB or LONG VARCHAR.
* fn:upper-case	COALESCE(UPPER({0}), '') COALESCE is not used if at compile-time it is determined that input can never be empty (NULL).	Argument is not of type CLOB or LONG VARCHAR.
fn:contains, fn:starts-with, fn:ends-with	LIKE with the ESCAPE clause and '|' as the escape character.	The first argument is not of type CLOB or LONG VARCHAR. The second argument is SQL constant or parameter.
fn:year-from-dateTime, fn:year-from-date	EXTRACT(YEAR FROM {0})	
fn:month-from-dateTime fn:month-from-date	EXTRACT(MONTH FROM {0})	
fn:day-from-dateTime fn:day-from-date	EXTRACT(DAY FROM {0})	
fn:hours-from-dateTime, fn:hours-from-time	EXTRACT(HOUR FROM {0})	
fn:minutes-from-dateTime, fn:minutes-from-time	EXTRACT(MINUTE FROM {0})	
fn:seconds-from-dateTime, fn:seconds-from-time	CAST(EXTRACT(SECOND FROM {0}) AS DECIMAL)	
op:hexBinary-equal	{0} = {1}	
op-bea:hexBinary-not-equal	{0} != {1}	
* Functions for which the default SQL syntax depends on the `default-syntax-for-empty-input` setting.		

Table 10-5 default-syntax-for-empty-input = strict-coalesce (Continued)

XQuery function	Default SQL syntax	Pushdown requirements
fn:empty	{0} IS NULL	
fn:exists	{0} IS NOT NULL (or as EXISTS if subqueries in the WHERE clause are supported)	
fn:count	COUNT (with COUNT DISTINCT support)	
* fn:sum	COALESCE(SUM({0}), 0) COALESCE is not used if at compile-time it is determined that input can never be empty (NULL).	
fn:min	MIN({0})	
fn:max	MAX({0})	
fn:avg	AVG({0})	
fn-bea:sql-like($str, $pattern)	{0} LIKE {1}	Arguments are not of type CLOB or LONG VARCHAR.
fn-bea:sql-like($str, $pattern, $escape)	{0} LIKE {1} ESCAPE {2}	Arguments are not of type CLOB or LONG VARCHAR.
fn-bea:left	LEFT({0}, {1})	First argument is not of type CLOB or LONG VARCHAR.
fn-bea:right	RIGHT({0}, {1})	First argument is not of type CLOB or LONG VARCHAR.
fn-bea:trim	LTRIM(RTRIM({0}))	Argument is not of type CLOB or LONG VARCHAR.
fn-bea:trim-left	LTRIM({0})	Argument is not of type CLOB or LONG VARCHAR.
* Functions for which the default SQL syntax depends on the `default-syntax-for-empty-input` setting.		

Table 10-5 default-syntax-for-empty-input = strict-coalesce (Continued)

XQuery function	Default SQL syntax	Pushdown requirements
fn-bea:trim-right	RTRIM({0})	Argument is not of type CLOB or LONG VARCHAR.
fn-bea:repeat	REPEAT({0})	Argument is not of type CLOB or LONG VARCHAR.
* Functions for which the default SQL syntax depends on the `default-syntax-for-empty-input` setting.		

Table 10-6 default-syntax-for-empty-input = strict

XQuery function	Default SQL syntax	Pushdown requirements
op:numeric-add	{0} + {1}	
op:numeric-multiply	{0} * {1}	
op:numeric-divide	{0} / {1}	
op:numeric-mod	MOD({0}, {1})	
fn:abs	ABS({0})	
fn:ceiling	CEILING({0})	
fn:floor	FLOOR({0})	
fn:round	FLOOR({0} + 0.5)	
fn-bea:sql-round	ROUND({0})	
* fn:concat	{0} || {1} || … || {n}	Arguments are not of type CLOB or LONG VARCHAR. Arguments must be non–nullable (as detected by the compiler).
* Functions for which the default SQL syntax depends on the `default-syntax-for-empty-input` setting.		

Table 10-6 default-syntax-for-empty-input = strict

XQuery function	Default SQL syntax	Pushdown requirements
* fn:substring ($str, $pos)	if $pos is a subtype of xs:integer SUBSTRING({0} FROM {1}) else SUBSTRING({0} FROM CAST({1}+0.5 AS INTEGER))	First argument is not of type CLOB or LONG VARCHAR. First argument must be non–nullable (as detected by the compiler).
* fn:substring($str, $pos, $len)	if $pos and $len are subtypes of xs:integer SUBSTRING({0} FROM {1} FOR {2}) else SUBSTRING({0} FROM CAST({1}+0.5 AS INTEGER) FOR CAST({2}+0.5 AS INTEGER))	First argument is not of type CLOB or LONG VARCHAR. First argument must be non–nullable (as detected by the compiler).
* fn:string-length	CHAR_LENGTH({0})	Argument is not of type CLOB or LONG VARCHAR. Argument must be non–nullable (as detected by the compiler).
* fn:lower-case	LOWER({0})	Argument is not of type CLOB or LONG VARCHAR Argument must be non–nullable (as detected by the compiler).
* fn:upper-case	UPPER({0})	Argument is not of type CLOB or LONG VARCHAR Argument must be non–nullable (as detected by the compiler).
fn:contains, fn:starts-with, fn:ends-with	LIKE with the ESCAPE clause and '|' as escape character.	The first argument is not of type CLOB or LONG VARCHAR. The second argument is SQL constant or parameter.
* Functions for which the default SQL syntax depends on the `default-syntax-for-empty-input` setting.		

Table 10-6 default-syntax-for-empty-input = strict

XQuery function	Default SQL syntax	Pushdown requirements
fn:year-from-dateTime, fn:year-from-date	EXTRACT(YEAR FROM {0})	
fn:month-from-dateTime fn:month-from-date	EXTRACT(MONTH FROM {0})	
fn:day-from-dateTime fn:day-from-date	EXTRACT(DAY FROM {0})	
fn:hours-from-dateTime, fn:hours-from-time	EXTRACT(HOUR FROM {0})	
fn:minutes-from-dateTime, fn:minutes-from-time	EXTRACT(MINUTE FROM {0})	
fn:seconds-from-dateTime, fn:seconds-from-time	CAST(EXTRACT(SECOND FROM {0}) AS DECIMAL)	
op:hexBinary-equal	{0} = {1}	
op-bea:hexBinary-not-equal	{0} != {1}	
fn:empty	{0} IS NULL	
fn:exists	{0} IS NOT NULL (or as EXISTS if subqueries in the WHERE clause are supported)	
fn:count	COUNT (with COUNT DISTINCT support)	
* `fn:sum`	SUM({0})	Argument must be non–nullable (as detected by the compiler).
fn:min	MIN({0})	
fn:max	MAX({0})	
* Functions for which the default SQL syntax depends on the `default-syntax-for-empty-input` setting.		

Table 10-6 default-syntax-for-empty-input = strict

XQuery function	Default SQL syntax	Pushdown requirements
fn:avg	AVG({0})	
fn-bea:sql-like($str, $pattern)	{0} LIKE {1}	Arguments are not of type CLOB or LONG VARCHAR.
fn-bea:sql-like($str, $pattern, $escape)	{0} LIKE {1} ESCAPE {2}	Arguments are not of type CLOB or LONG VARCHAR.
fn-bea:left	LEFT({0}, {1})	First argument is not of type CLOB or LONG VARCHAR.
fn-bea:right	RIGHT({0}, {1})	First argument is not of type CLOB or LONG VARCHAR.
fn-bea:trim	LTRIM(RTRIM({0}))	Argument is not of type CLOB or LONG VARCHAR.
fn-bea:trim-left	LTRIM({0})	Argument is not of type CLOB or LONG VARCHAR.
fn-bea:trim-right	RTRIM({0})	Argument is not of type CLOB or LONG VARCHAR.
fn-bea:repeat	REPEAT({0})	Argument is not of type CLOB or LONG VARCHAR.
* Functions for which the default SQL syntax depends on the `default-syntax-for-empty-input` setting.		

Table 10-7 default-syntax-for-empty-input = lax

XQuery function	Default SQL syntax	Pushdown requirements
op:numeric-add	{0} + {1}	
op:numeric-multiply	{0} * {1}	
* Functions for which the default SQL syntax depends on the `default-syntax-for-empty-input` setting.		

Table 10-7 default-syntax-for-empty-input = lax

XQuery function	Default SQL syntax	Pushdown requirements
op:numeric-divide	{0} / {1}	
op:numeric-mod	MOD({0}, {1})	
fn:abs	ABS({0})	
fn:ceiling	CEILING({0})	
fn:floor	FLOOR({0})	
fn:round	FLOOR({0} + 0.5)	
fn-bea:sql-round	ROUND({0})	
* fn:concat	{0} || {1} || … || {n}	Arguments are not of type CLOB or LONG VARCHAR.
* fn:substring ($str, $pos)	if $pos is a subtype of xs:integer SUBSTRING({0} FROM {1}) else SUBSTRING({0} FROM CAST({1}+0.5 AS INTEGER))	First argument is not of type CLOB or LONG VARCHAR.
* fn:substring($str, $pos, $len)	if $pos and $len are subtypes of xs:integer SUBSTRING({0} FROM {1} FOR {2}) else SUBSTRING({0} FROM CAST({1}+0.5 AS INTEGER) FOR CAST({2}+0.5 AS INTEGER))	First argument is not of type CLOB or LONG VARCHAR.
* Functions for which the default SQL syntax depends on the `default-syntax-for-empty-input` setting.		

Table 10-7 default-syntax-for-empty-input = lax

XQuery function	Default SQL syntax	Pushdown requirements
* fn:string-length	CHAR_LENGTH({0})	Argument is not of type CLOB or LONG VARCHAR.
* fn:lower-case	LOWER({0})	Argument is not of type CLOB or LONG VARCHAR.
* fn:upper-case	UPPER({0})	Argument is not of type CLOB or LONG VARCHAR.
fn:contains, fn:starts-with, fn:ends-with	LIKE with ESCAPE clause and '|' as escape character	The first argument is not of type CLOB or LONG VARCHAR. The second argument is SQL constant or parameter.
fn:year-from-dateTime, fn:year-from-date	EXTRACT(YEAR FROM {0})	
fn:month-from-dateTime fn:month-from-date	EXTRACT(MONTH FROM {0})	
fn:day-from-dateTime fn:day-from-date	EXTRACT(DAY FROM {0})	
fn:hours-from-dateTime, fn:hours-from-time	EXTRACT(HOUR FROM {0})	
fn:minutes-from-dateTime, fn:minutes-from-time	EXTRACT(MINUTE FROM {0})	
fn:seconds-from-dateTime, fn:seconds-from-time	CAST(EXTRACT(SECOND FROM {0}) AS DECIMAL)	
op:hexBinary-equal	{0} = {1}	
op-bea:hexBinary-not-equal	{0} != {1}	
fn:empty	{0} IS NULL	

* Functions for which the default SQL syntax depends on the `default-syntax-for-empty-input` setting.

Table 10-7 default-syntax-for-empty-input = lax

XQuery function	Default SQL syntax	Pushdown requirements
fn:exists	{0} IS NOT NULL (or as EXISTS if subqueries in the WHERE clause are supported)	
fn:count	COUNT (with COUNT distinct support)	
* fn:sum	SUM({0})	
fn:min	MIN({0})	
fn:max	MAX({0})	
fn:avg	AVG({0})	
fn-bea:sql-like($str, $pattern)	{0} LIKE {1}	Arguments are not of type CLOB or LONG VARCHAR.
fn-bea:sql-like($str, $pattern, $escape)	{0} LIKE {1} ESCAPE {2}	Arguments are not of type CLOB or LONG VARCHAR.
fn-bea:left	LEFT({0}, {1})	First argument is not of type CLOB or LONG VARCHAR.
fn-bea:right	RIGHT({0}, {1})	First argument is not of type CLOB or LONG VARCHAR.
fn-bea:trim	LTRIM(RTRIM({0}))	Argument is not of type CLOB or LONG VARCHAR.
fn-bea:trim-left	LTRIM({0})	Argument is not of type CLOB or LONG VARCHAR.
fn-bea:trim-right	RTRIM({0})	Argument is not of type CLOB or LONG VARCHAR.
fn-bea:repeat	REPEAT({0})	Argument is not of type CLOB or LONG VARCHAR.
* Functions for which the default SQL syntax depends on the `default-syntax-for-empty-input` setting.		

Translating Built-In XQuery Operators Into SQL

The XQuery Functions and Operators specification defines built-in operators into which arithmetic and comparison operations are translated. For some operations ALDSP defines additional operators that it uses for evaluation. These additional operators can also be used for specifying XQuery to SQL translation.

Tip: For references to the XQuery specifications, see Supported XQuery Specifications in the *XQuery and XQSE Developer's Guide*.

For each of the following arithmetic operations, ALDSP defines more specific operations for the following types: integer, decimal, double, float. These specific operations can be used to specify a better type match when defining a SQL generation rule.

Tip: The function operation prefixes used in the expanded operations (such as op-bea) are discussed in "Standard and ALDSP Namespaces for Functions and Types" on page 10-50.

- op:numeric-add
- op:numeric-subtract
- op:numeric-multiply
- op:numeric-divide
- op:numeric-integer-divide
- op:numeric-mod

For example, the following four operations are defined for op:numeric-add:

- op-bea:integer-add
- op-bea:decimal-add
- op-bea:float-add
- op-bea:double-add

Comparison operations in the XQuery are defined by three operators:

- op:<*type*>-equals
- op:<*type*>-less-than
- op:<*type*>-greater-than

ALDSP adds three more operations for each type:

- op-bea:<*type*>-not-equals
- op-bea:<*type*>-less-than-or-equals
- op-bea:<*type*>-greater-than-or-equals

For numeric types, each operator op-bea:numeric-<*comparison_op*> is further expanded into four numeric types:

op-bea:integer-<*comparison_op*>, op-bea:decimal-<*comparison_op*>, op-bea:double-<*comparison_op*>, op-bea:float-<*comparison_op*>.

Additional numeric comparisons added by ALDSP follow the same pattern. For example

op-bea:numeric-not-equals is expanded into:

- op-bea:integer-not-equals
- op-bea:decimal-not-equals
- op-bea:double-not-equals
- op-bea:float-not-equals

All six string comparison operators are defined as ALDSP specific operators:

- op-bea:string-equals
- op-bea:string-less-than
- op-bea:string-greater-than
- op-bea:string-not-equals
- op-bea:string-less-than-or-equals
- op-bea:string-greater-than-or-equals

Standard and ALDSP Namespaces for Functions and Types

Table 10-8 lists the standard and ALDSP namespaces for functions and types. Table 10-9 lists and describes each of the type namespaces.

Table 10-8 Function and operators namespaces

Prefix	Namespace	Description
fn	http://www.w3.org/2004/07/xpath-functions	Standard XQuery functions
op	http://www.w3.org/2004/07/xpath-operators	Standard XQuery operators
fn-bea	http://www.bea.com/xquery/xquery-functions	ALDSP extension functions
op-bea	http://www.bea.com/xquery/xquery-operators	ALDSP extension operators

Table 10-9 Type namespaces

Prefix	Namespace	Description
xs	http://www.w3.org/2001/XMLSchema	XML Schema types
xdt	http://www.w3.org/2004/07/xpath-datatypes	Additional XQuery types
dt-bea	http://www.bea.com/xquery/xquery-datatypes	Additional ALDSP types. Currently only one: *dt-bea:numeric* (common numeric type)

Function and Type Name Resolution Process

The Relational Wrapper Extension Framework looks up functions, operators, and types by name as follows:

1. Attempt a lookup using the specified QName. If the object is found, return it.
2. If the namespace is empty or the prefix is not specified, loop through all commonly used namespaces for this object kind (see "Standard and ALDSP Namespaces for Functions and Types" on page 10-50) and try to find the object in each of these namespaces.

For example, suppose the following function definition exists:

<function name="round">ROUND({0})</function>

First, that name is resolved to a QName in the default element namespace and looked up. Suppose then that the XQuery function with this name is not found (for example, if there was no default namespace used in the XML document). Then the system will try start searching for the following functions (in this order): fn:round, op:round, fn-bea:round, op-bea:round. The system will find fn:round and register it with the specified SQL syntax.

A similar lookup process is applied for types when reading cast operation definitions. For types, the system automatically searches in xs, xdt and dt-bea namespaces.

Note that the `arity` attribute is also optional and only required to disambiguate between functions with the same name, for example, a substring with 2 and 3 arguments.

Abstract SQL Providers

ALDSP provides a group of three abstract base classes that provide functionality to the Configurable Relational Provider. The AbstractSQLProvider class is the default parent class of the Configurable Relational Provider. You can specify an abstract provider class in the Configurable Relational Provider's deployment descriptor with the `parent` element. See "Using the Configurable Relational Provider" on page 10-13.

This section discusses the abstract relational provider classes, and contains these sections:

- AbstractSQLProvider
- AbstractSQL89Provider
- AbstractSQL92Provider

AbstractSQLProvider

AbstractSQLProvider is an abstract base class. All other abstract and concrete relational provider classes extend this class. This class is used as a parent provider when the parent is not specified in the deployment descriptor of a provider; therefore, this class is not explicitly registered in the provider registry.

Table 10-10 summarizes the level of SQL support provided by AbstractSQLProvider:

Table 10-10 AbstractSQLProvider Features

Feature	Status
Standard JDBC datatypes	Supported
Trivial select-project queries (for example: `select ... from ... where`)	Supported
Joins, group by, and order by	Not supported
Catalogs and schemas when addressing tables	Not supported
Catalog, schema, and table quotes	Set to “empty string”
Catalog and schema separator	Set to ‘.’ (although separators are not used for queries generated by this provider)
Runtime properties	All set to `false`.

Table 10-11 lists the supported functions and operators for AbstractSQLProvider.

Table 10-11 Supported Functions and Operators for AbstractSQLProvider

XQuery function	SQL Syntax	Pushdown Requirements / Comments
and, or, fn:not	AND, OR, NOT	None.
op:numeric-equal op:numeric-less-than op:numeric-greater-than op-bea:numeric-less-than-or-equal op-bea:numeric-greater-than-or-equal op-bea:numeric-not-equal	=, <, >, <=, >=, !=	
op-bea:string-equal op-bea:string-less-than op-bea:string-greater-than op-bea:string-less-than-or-equal op-bea:string-greater-then-or-equal op-bea:string-not-equal	=, <, >, <=, >=, !=	Both arguments are not CLOB or LONG VARCHAR
op:dateTime-equal op:dateTime-less-than op:dateTime-greater-than op-bea:dateTime-less-than-or-equal op-bea:dateTime-greater-than-or-equal op-bea:dateTime-not-equal	=, <, >, <=, >=, !=	None.
op:date-equal op:date-less-than op:date-greater-than op-bea:date-less-than-or-equal op-bea:date-greater-than-or-equal op-bea:date-not-equal	=, <, >, <=, >=, !=	None.

Table 10-11 Supported Functions and Operators for AbstractSQLProvider (Continued)

XQuery function	SQL Syntax	Pushdown Requirements / Comments
op:time-equal op:time-less-than op:time-greater-than op-bea:time-less-than-or-equal op-bea:time-greater-than-or-equal op-bea:time-not-equal	=, <, >, <=, >=, !=	None.
op:hexBinary-equal op-bea:hexBinary-not-equal	=, !=	Only if both arguments are BINARY or VARBINARY.

AbstractSQL89Provider

AbstractSQL89Provider extends AbstractSQLProvider (see "AbstractSQLProvider" on page 10-52). This class adds support for additional clauses, functions, and updates. The AbstractSQL89Provider class includes these features:

- Supports SQL89-style inner joins (for example, `select … from A,B where A.<x> = B.<x>`).
- Supports order by column (null order is assumed to be 'low').
- Supports group by column (and aggregate functions).
- Schemas are used for table addressing (using dot as a separator).
- Supports subqueries in where clause.

Table 10-12 lists the supported functions and operators for AbstractSQL89Provider. These functions and operators are in addition to the ones provided by the parent class, AbstractSQLProvider.

Table 10-12 Supported Functions and Operators for AbstractSQL89Provider

XQuery function	SQL Syntax	Pushdown Requirements / Comments
op:numeric-add op:numeric-subtract op:numeric-multiply op:numeric-divide (except op-bea:integer-divide)	+, -, *, /	None.
fn:exists	{0} IS NOT NULL (EXISTS in the WHERE clause is not supported)	None.
fn:empty	{0} IS NULL	None.
fn:count	COUNT (with COUNT DISTINCT support)	None.
fn:sum	SUM({0})	Note that this function does not match XQuery semantics. For empty (NULL) input, the function returns empty (NULL) instead of 0. XQuery specifies that SUM(())=0; where () is an empty sequence. This provider translates the function to SQL as SUM(...). However, in SQL, SUM(NULL)=NULL, which is equivalent to () in XQuery.
fn:min	MIN({0})	None.
fn:max	MAX({0})	None.
fn:avg	AVG({0})	None.

Table 10-12 Supported Functions and Operators for AbstractSQL89Provider (Continued)

XQuery function	SQL Syntax	Pushdown Requirements / Comments
fn-bea:sql-like($str, $pattern)	{0} LIKE {1}	First argument is not CLOB or LONG VARCHAR. Second (and third) arguments are a SQL constant or parameter.
fn-bea:sql-like($str, $pattern, $escape)	{0} LIKE {1} ESCAPE {2}	None.

AbstractSQL92Provider

AbstractSQL92Provider extends AbstractSQL89Provider (see "AbstractSQL89Provider" on page 10-54). This class adds support for SQL92-style joins (inner and outer), subqueries, and other features. The AbstractSQL92Provider class supports:

- Inner and outer-joins
- Subqueries in from clause
- Order by and group by expression
- Case expressions
- Updates (update/identity-fetch – JDBC kind)

Table 10-13 lists the supported functions and operators for AbstractSQL92Provider. These functions and operators are in addition to the ones provided by the parent class, AbtractSQL89Provider.

Table 10-13 Supported Functions and Operators for AbstractSQL92Provider

XQuery function	SQL Syntax	Pushdown Requirements / Comments
fn:concat	COALESCE({0} || {1} || … || {n}, '') COALESCE is not used if at compile-time it is determined that input can never be empty (NULL).	Arguments are not of type CLOB or LONG VARCHAR.
fn:upper-case	COALESCE(UPPER({0},'') COALESCE is not used if at compile-time it is determined that input can never be empty (NULL).	Argument is not of type CLOB or LONG VARCHAR.
fn:lower-case	COALESCE(LOWER({0},'') COALESCE is not used if at compile-time it is determined that input can never be empty (NULL).	Argument is not of type CLOB or LONG VARCHAR.
fn:substring ($str, $pos)	if $pos is a subtype of xs:integer COALESCE(SUBSTRING({0} FROM {1}), '') else COALESCE(SUBSTRING({0} FROM CAST({1}+0.5 AS INTEGER)), '') COALESCE is not used if at compile-time it is determined that input can never be empty (NULL).	First argument is not of type CLOB or LONG VARCHAR.

Table 10-13 Supported Functions and Operators for AbstractSQL92Provider (Continued)

XQuery function	SQL Syntax	Pushdown Requirements / Comments
fn:substring($str, $pos, $len)	if $pos and $len are subtypes of xs:integer COALESCE(SUBSTRING({0} FROM {1} FOR {2}), '') else COALESCE(SUBSTRING({0} FROM CAST({1}+0.5 AS INTEGER) FOR CAST({2}+0.5 AS INTEGER)), '') COALESCE is not used if at compile-time it is determined that input can never be empty (NULL).	First argument is not of type CLOB or LONG VARCHAR.
fn:string-length	COALESCE(CHAR_LENGTH({0}), 0) COALESCE is not used if at compile-time it is determined that input can never be empty (NULL).	Argument is not of type CLOB or LONG VARCHAR.
fn:contains, fn:starts-with, fn:ends-with	LIKE with ESCAPE clause and '|' as escape character	The first argument is not of type CLOB or LONG VARCHAR. The second argument is SQL constant or parameter.
fn:year-from-dateTime, fn:year-from-date	EXTRACT(YEAR FROM {0})	
fn:month-from-dateTime fn:month-from-date	EXTRACT(MONTH FROM {0})	
fn:day-from-dateTime fn:day-from-date	EXTRACT(DAY FROM {0})	
fn:hours-from-dateTime, fn:hours-from-time	EXTRACT(HOUR FROM {0})	

Table 10-13 Supported Functions and Operators for AbstractSQL92Provider (Continued)

XQuery function	SQL Syntax	Pushdown Requirements / Comments
fn:minutes-from-dateTime, fn:minutes-from-time	EXTRACT(MINUTE FROM {0})	
fn:seconds-from-dateTime, fn:seconds-from-time	CAST(EXTRACT(SECOND FROM {0}) AS DECIMAL)	
fn:sum	COALESCE(SUM({0}), 0) COALESCE is not used if at compile-time it is determined that input can never be empty (NULL). SUM(DISTINCT …) is supported	
fn:min	MIN(DISTINCT …) supported	
fn:max	MAX(DISTINCT …) supported	
fn:avg	AVG(DISTINCT …) supported	
fn-bea:left	SUBSTRING({0} FROM 1 FOR {1})	Argument is not of type CLOB or LONG VARCHAR.
fn-bea:trim	TRIM (BOTH ‘ ‘ FROM {0})	Argument is not of type CLOB or LONG VARCHAR.
fn-bea:trim-left	TRIM(LEADING ‘ ‘ FROM {0})	Argument is not of type CLOB or LONG VARCHAR.
fn-bea:trim-right	TRIM(TRAILING‘ ‘ FROM {0})	Argument is not of type CLOB or LONG VARCHAR.
fn-bea:date-from-dateTime	CAST({0} AS DATE)	

Table 10-14 lists the cast operations that are pushed down by AbstractSQL92Provider.

Table 10-14 Supported Cast Operations for AbstractSQL92Provider

Source Type	Target Type	SQL Syntax	Comments
subtypes of xs:int	xs:string	CAST({0} AS VARCHAR(11))	
xs:string	xs:double	CAST({0} AS DOUBLE PRECISION)	Argument is not of type CLOB or LONG VARCHAR.
subtypes of numeric	xs:double		
xs:string	xs:float	CAST({0} AS REAL)	Argument is not of type CLOB or LONG VARCHAR.
subtypes of numeric	xs:float		
xs:string	xs:int	CAST({0} AS INT)	Argument is not of type CLOB or LONG VARCHAR.
subtypes of numeric	xs:int		
xs:string	xs:short	CAST({0} AS SMALLINT)	Argument is not of type CLOB or LONG VARCHAR.
subtypes of numeric	xs:short		
xs:dateTime	xs:date	CAST({0} AS DATE)	