BEA AquaLogic Data Services Platform™

Samples Tutorial

Note: In some cases illustrations, directories, and paths reference Liquid Data ("ld"), the original name of the Data Services Platform.

Product version: 2.5
Document Date: June 2005
Revised: March 2007

Copyright

Restricted Rights Legend

Trademarks and Service Marks

About This Document

Welcome to the AquaLogic Data Services Platform Samples Tutorial. In this document, you are provided with step-by-step instructions that show how you can solve many of the types of data integration problems frequently faced by Information Technology (IT) managers and staff. These issues include:

- What is the best way to normalize data drawn from widely divergent sources?
- Having normalized the data, can you access it, ideally through a single point of access?
- After you define a single point of access, can you develop reusable queries that are easily tested, stored, and retrieved?
- After you develop your query set, can you easily incorporate results into widely available applications?

Other questions may occur. Is the data-rich solution scalable? Is it reusable throughout the enterprise? Are the original data sources largely transparent to the application — or do they become an issue each time you want to make a minor adjustments to queries or underlying data sources?

Document Organization

This guide is organized into 35 tutorials that illustrate many aspects of Data Services Platform functionality:

- **Data service development.** In which you specify the query functions that you will use to access, aggregate, and transform distributed, disparate data into a unified view. In this stage, you also specify the XML type that defines the data view that will be available to client-side applications.

- **Data modeling**. In which you define a graphical representation of data resource relationships and functions.
- **Client-side development.** In which you define an environment for retrieving data results.

Each tutorial consists of an overview plus lessons that demonstrate AquaLogic Data Services Platform capabilities on a topic-by-topic basis. Each tutorial is structured as a series of procedural steps that details the specific actions needed to complete that part of the demonstration.

Note: The tutorials build on each other and must be completed in sequential order. Unless a step or lesson is labeled as optional it should be completed. Otherwise you may not be able to successfully complete a subsequent, dependent lesson.

Technical Prerequisites

The lessons within this guide require a familiarity with the following topics: data integration and aggregation concepts, the BEA WebLogic® Platform™ (particularly WebLogic Server and WebLogic Workshop), Java, query concepts, and the environment in which you will install and use AquaLogic Data Services Platform.

For some lessons, a background in XQuery is helpful.

System Requirements

To complete the lessons, your computer requires:

Server	BEA WebLogic Server 8.1 Service Pack 5
Domain	dplatform
Application	BEA AquaLogic Data Services Platform 2.5
Operating System	Windows 2000 or Windows XP
Memory	512 MB RAM minimum; 1 GB RAM recommende
Browser	Internet Explorer 6 or higher or equivilent

Data Sources Used Within These Tutorials

The Samples Tutorial builds data services that draw on a variety of underlying data sources. These data sources, which are provided with the product, are described in the following table:

Data Source Type	Data Source	Data
Relational	Customer Relationship Management (CRM) RTLCUSTOMER database	Customer and credit card data
Relational	Order Management System (OMS) RTLAPPLOMS database	Apparel product, order, and order line data
Relational	Order Management System (OMS) RTLELECOMS database	Electronics product, order, and order line data
Relational	RTLSERVICE database	Customer service data, organized in a single Service Case table
Web service	CreditRatingWS	Credit rating data
Stored procedure	GETCREDITRATING_SP	Customer credit rating information
Java function	Functions.DSML	Java function enabling LDAP access
Java function	Functions.excel_jcom	Excel spreadsheet data, via JCOM
Java function	Functions.CreditCardClient	Customer credit card information, via an XMLBean
XML files	ProductUNSPSC.xsd	Third-party product information
Flat file	Valuation.csv	Data received from an internal department that deals with customer scoring and valuation model

Related Information

In addition to the material covered in this guide, you may want to review the wealth of resources available at the BEA Web site, WebLogic developer site, and third-party sites. Information at these sites includes datasheets, product brochures, customer testimonials, product documentation, code samples, white papers, and more.

For more information about Java and XQuery, refer to the following sources:

- The Sun Microsystems, Inc. Java site at:

http://java.sun.com/

- The World Wide Web Consortium XML Query section at:

http://www.w3.org/XML/Query

For more information about BEA products, refer to the following sources:

- ALDSP documentation site at:

http://edocs.bea.com/aldsp/docs25/

- BEA e-docs documentation site at:

http://e-docs.bea.com/

- BEA online community for WebLogic developers at:

http://dev2dev.bea.com

Core (Tutorials 1-17)

BEA AquaLogic Data Services Platform approaches the problem of creating integration architectures by providing tools that let you build physical data services around individual physical data sources, and then develop logical data services and business logic that integrate and return data from multiple physical and logical data services. Logical data services use easily-maintained, graphically-designed XML queries (XQueries) to access, aggregate, transform, and deliver its data results.

Developing ALDSP services involves three basic steps:

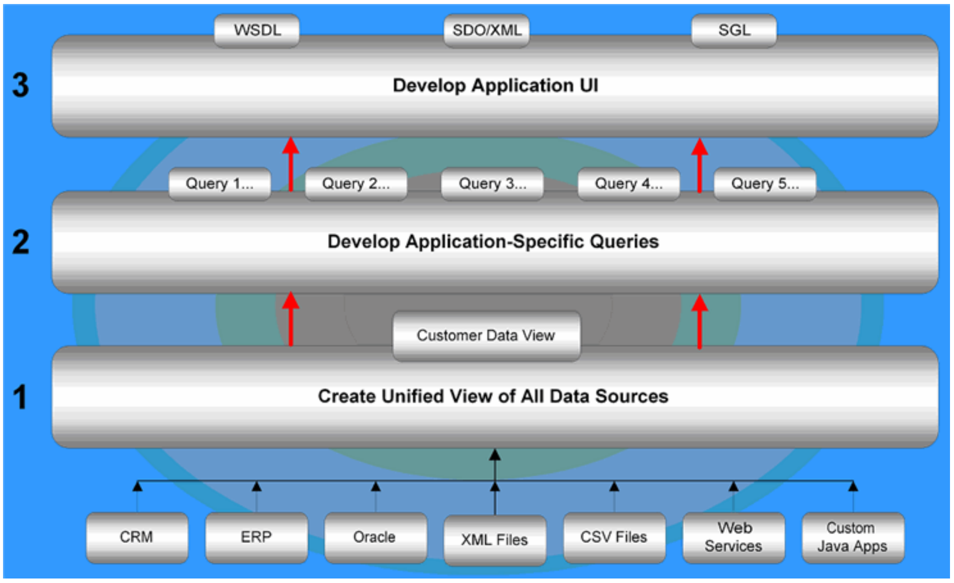
1. **Create a unified view of information from all relevant sources.** This step, which involves development of physical data services and (optionally) data models, is typically performed by a data services architect who understands the information available in underlying sources and can define the unified view that different projects will use. ALDSP is capable of modeling relational

and non-relational sources; it includes tools for introspection and mapping of the underlying sources to the unified data view.

2. **Develop application-specific queries.** This step, which involves development of logical data services, is typically performed by application developers who write simple queries against the unified view to get the required data. ALDSP provides tools to visually create robust XQueries and also publish them as services.

3. **Tie query results to client applications.** This step, which involves accessing data through a variety of consuming applications, is typically performed by application developers who execute the queries and receive results as XML or Java objects. In addition, ALDSP provides an out-of-the-box Workshop control to easily develop portal or Web applications from which to access data retrieved by a data service.

Figure 0-1 Data Services Platform Development Process



Data Services Platform Development Process

As part of the development process, ALDSP provides flexible options for updating both relational and non-relational data sources. ALDSP lets you write update logic via an EJB in BEA WebLogic Server™; via a database, JMS, or Data Services Platform Control in Workshop; or via a business process in BEA WebLogic Integration™.

In addition, ALDSP provides visual tools for managing various administrative tasks, including controlling data service metadata, caching, and security.

The initial 17 tutorials illustrate ALDSP's most commonly used capabilities: developing and testing physical and logical data services, accessing data services through various consuming applications, updating underlying data sources, and managing various administrative tasks.

Note: The lessons build upon one another and should be completed in sequential order.

Advanced (Tutorials 18-35)

In advanced totorials you will build upon that knowledge to:

- Build queries in both XQuery Editor View and Source View.
- Create models for logical data services.
- Run ad hoc queries.
- Use update overrides to perform custom data manipulations, update Web services, and overwrite SQL updates.
- Use the automatically generated Query Plan.
- Re-use XQuery code.
- Configure alternative sources for unavailable data sources.
- Use SQL Exits to enable retrieving data from an SQL statement.
- Enable fine-grained caching.
- Enable element-level security.
- Create data services from stored procedures, Java functions, XML files, and flat files.
- Create an XQuery function library.

2/9 14:10

Introducing the Data Services Environment

BEA AquaLogic Data Services Platform provides the tools and components that let you build physical data services around individual physical data sources, and then develop the logical data services and business logic that integrate data from multiple physical and logical data services. The environment also lets you test the data service and manage data service metadata, caching, and security.

The basic menus, behavior, and look-and-feel associated with the WebLogic Workshop environment apply to ALDSP. However, there are several tools and components within WebLogic Workshop that are especially relevant to ALDSP. In this lesson, you will learn about a few of those tools and components. In addition, you will learn how to complete several basic tasks, such as starting and stopping WebLogic Server, that are essential to using WebLogic Workshop.

As the first lesson within the AquaLogic Data Services Platform Samples Tutorial, there are no dependencies on other lessons. However, your familiarity with WebLogic Workshop is assumed. Workshop is fully described in online documentation, which you can view at:

`http://edocs.bea.com/workshop/docs81/index.html`

Objectives

After completing this exercise, you will be able to:

- Navigate the ALDSP environment.
- Start and stop WebLogic Server.
- Save a Data Services application and associated files.

Overview

WebLogic Workshop consists of two parts: an Integrated Development Environment (IDE) and a standards-based runtime environment. The purpose of the IDE is to remove the complexity in building applications for the entire WebLogic platform. Applications you build in the IDE are constructed from high-level components rather than low-level API calls. Best practices and productivity are built into both the IDE and runtime.

1.1 Starting WebLogic Workshop

The first step is starting WebLogic Workshop and opening the RTLApp sample application, which you will use in the next lesson.

Objectives

In this exercise, you will:

- Start WebLogic Workshop.
- Open the RTLApp application.

Instructions

1. Choose Start → Programs → BEA WebLogic Platform 8.1 → Examples → WebLogic Workshop → Start Workshop with Sample Applications.

   If this is the first time you are starting WebLogic Workshop, then the SamplesApp project opens. Otherwise, the project that you last opened appears.

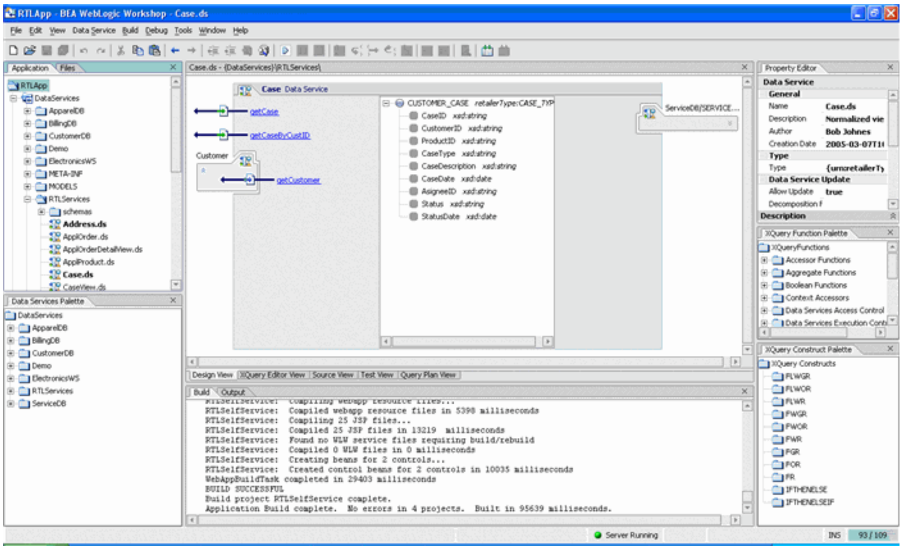
2. Choose File → Open → Application
3. Open the `RTLApp.work` file from the following location:

```
<beahome>\weblogic81\samples\liquiddata\RTLApp\
```

Note: Depending on your computer settings, the `.work` extension may not be visible.

In Listing 1-1 RTLApp in Design View for `Case.ds`, the RTLApp application opens in Design View for the Case data service. If this is not the view that you see, double-click `Case.ds` located at DataServices/RTLServices and select the Design View tab.

Figure 1-1 RTLApp in Design View for Case.ds



Name	Case.ds
Description	Normalized vie
Author	Bob Johnes
Creation Date	2005-03-07T1
Type	{urn:retailerTy
Allow Update	true

```
RTLSelfService:  Compiled webapp resource files in 5398 milliseconds
RTLSelfService:  Compiling 25 JSP files...
RTLSelfService:  Compiled 25 JSP files in 13219  milliseconds
RTLSelfService:  Found no WLW service files requiring build/rebuild
RTLSelfService:  Compiled 0 WLW files in 0 milliseconds
RTLSelfService:  Creating beans for 2 controls...
RTLSelfService:  Created control beans for 2 controls in 10035 milliseconds
WebAppBuildTask completed in 29403 milliseconds
BUILD SUCCESSFUL
Build project RTLSelfService complete.
Application Build complete.  No errors in 4 projects.  Built in 95639 milliseconds.
```

Note: The RTLApp application opens in the last active view. This action also resets the default WebLogic server home directory instance to the ldplatform sample domain.

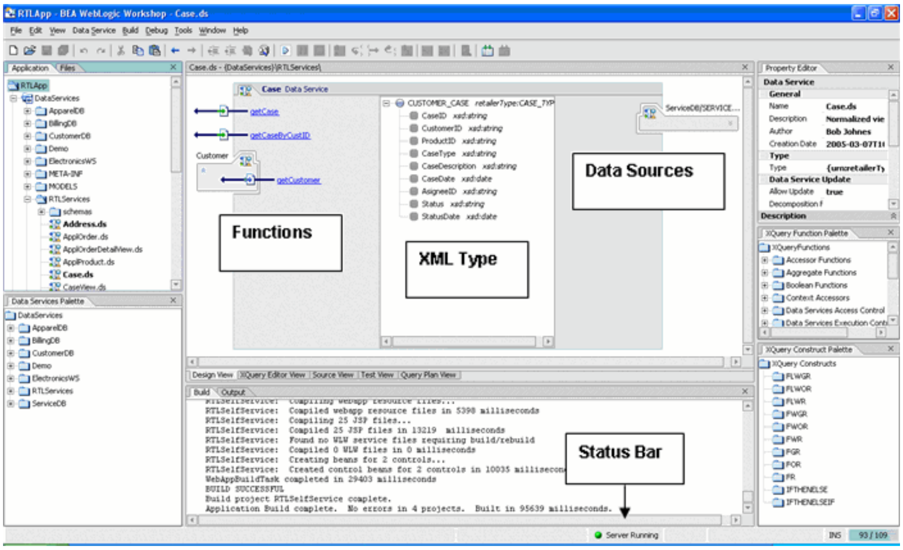
1.2 Navigating the ALDSP Integrated Development Environment (IDE)

Within the WebLogic Workshop environment, there are several tools and components that are relevant to developing ALDSP applications and projects. Five of the most frequently used are:

- Application Pane
- Design View
- XQuery Editor View
- Source View
- Test View

Screenshots of the environment are taken from within the RTLApp application.

Figure 1-2 Data Services Platform Running in WebLogic Workshop



```
RTLSelfService:  Compiled webapp resource files in 5398 milliseconds
RTLSelfService:  Compiling 25 JSP files...
RTLSelfService:  Compiled 25 JSP files in 13219  milliseconds
RTLSelfService:  Found no WLW service files requiring build/rebuild
RTLSelfService:  Compiled 0 WLW files in 0 milliseconds
RTLSelfService:  Creating beans for 2 controls...
RTLSelfService:  Created control beans for 2 controls in 10035 milliseconds
WebAppBuildTask completed in 29403 milliseconds
BUILD SUCCESSFUL
Build project RTLSelfService complete.
Application Build complete.  No errors in 4 projects.  Built in 95639 milliseconds.
```

Objectives

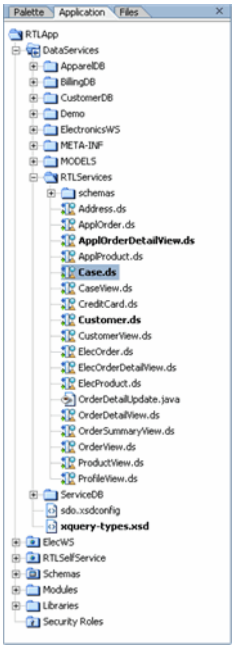
In this exercise, you will:

- Explore five of the most frequently used development tools.
- Discover the features and functions of these tools.

Application Pane

The Application pane displays a hierarchical representation of a ALDSP application.

Figure 1-3 Application Pane



```
RTLApp
  DataServices
    ApparelDB
    BillingDB
    CustomerDB
    Demo
    ElectronicsWS
    META-INF
    MODELS
    RTLServices
      schemas
      Address.ds
      ApplOrder.ds
      ApplOrderDetailView.ds
      ApplProduct.ds
      Case.ds
      CaseView.ds
      CreditCard.ds
      Customer.ds
      CustomerView.ds
      ElecOrder.ds
      ElecOrderDetailView.ds
      ElecProduct.ds
      OrderDetailUpdate.java
      OrderDetailView.ds
      OrderSummaryView.ds
      OrderView.ds
      ProductView.ds
      ProfileView.ds
    ServiceDB
    sdo.xsdconfig
    xquery-types.xsd
  ElecWS
  RTLSelfService
  Schemas
  Modules
  Libraries
  Security Roles
```

A Workshop application is a collection of all resources and components—projects, schemas, modules, libraries, and security roles—deployed as a unit to an instance of WebLogic Server. Only one application can be active at a time. Open files display in boldface type.

If the Application pane is not open, complete one of the following options:

1. Choose View → Application.
2. Press Alt+1.

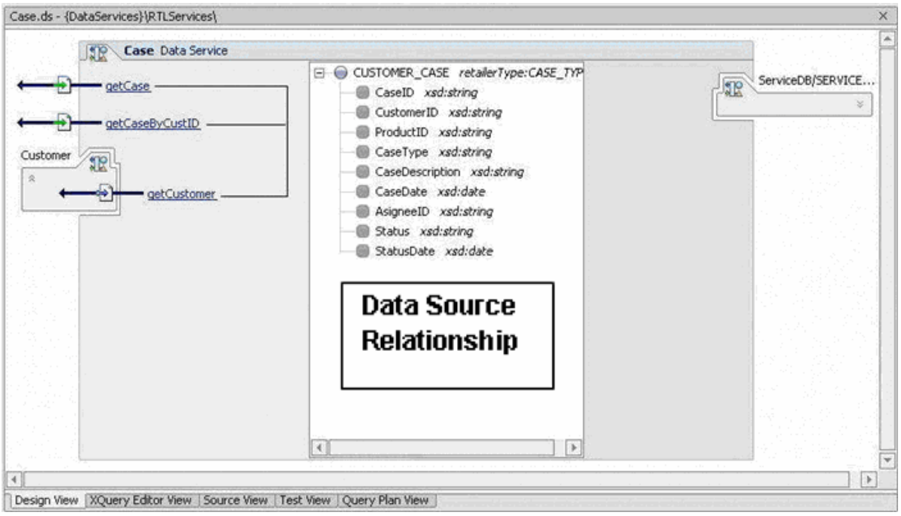
Design View

Design View presents an editable, graphical representation of a data service. It is a single point of consolidation for a data service's query functions and other business logic. Using Design View, you can:

- View the data service's XML type, native data types, functions, and data source relationships.
- Add functions and data source relationships.
- Create an XML type definition for elements within the data service, such as xs:string or xs:date.

- Associate the data service with an XML Schema Definition (.xsd) that defines the unified view for all retrieved data.

Figure 1-4 Design View of a Logical Data Service



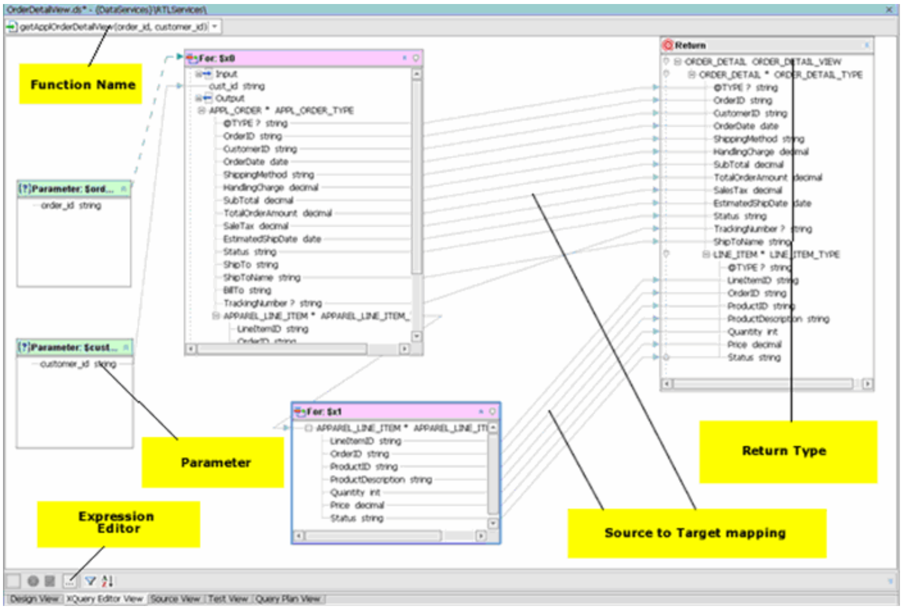
If Design View is not open, complete the following steps:

1. Open a data service such as `Case.ds` located in DataServices/RTLServices.
2. Select the Design View tab.

XQuery Editor View

XQuery Editor View provides a graphical, drag-and-drop approach to constructing queries. Using this view, you can inspect or edit the query Return type and add the data source nodes, parameters, expressions, conditions, and source-to-target mappings that comprise data service query functions.

Figure 1-5 Sample XQuery Editor View



If XQuery Editor View is not open:

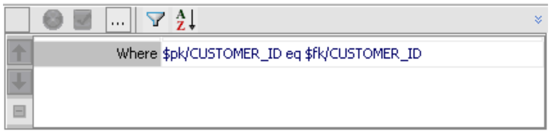
1. Open a data service such as `Case.ds` located in DataServices/RTLServices
2. Select the XQuery Editor View tab.

XQuery Editor View Tools

XQuery Editor View includes several editors and palettes that simplify the construction of queries:

- **Expression Editor.** Lets you add where and order by conditions to let or for nodes. The Expression Editor is only active when you click on the specific node header.

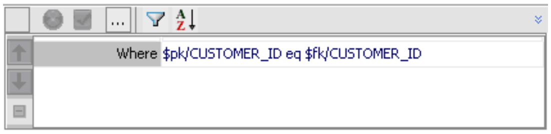
Figure 1-6 Expression Editor



Where $pk/CUSTOMER_ID eq $fk/CUSTOMER_ID

- **Data Services Palette**. Lets you add previously-defined query functions as data sources. Each function displays as a for node, which serves as a for clause within the FLWOR (for-let-where-order by-return) statement that is the heart of an XQuery.

Figure 1-7 Data Services Palette



Where $pk/CUSTOMER_ID eq $fk/CUSTOMER_ID

To add data sources, drag and drop an item from the Data Services Palette into the XQuery Editor View work area. After you drop the node into XQuery Editor View, the node's data source schema (shape) displays in the XQuery Editor View.

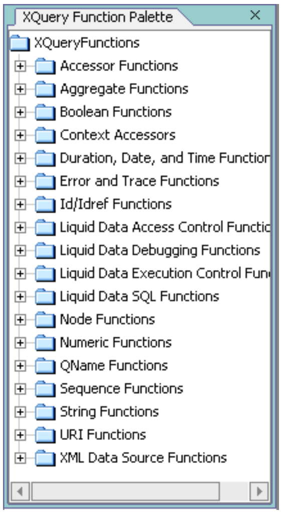
If the Data Services Palette is not open, choose View → Windows → Data Services Palette.

- **XQuery Function Palette.** Lets you add any of the more than 100 built-in functions provided within the XQuery language. In addition, you can add any of the special built-in functions defined by BEA.

To add a built-in function, drag and drop the selected item into the Expression Editor.

If XQuery Function Palette is not open, choose View → Windows → XQuery Function Palette.

Figure 1-8 XQuery Function Palette



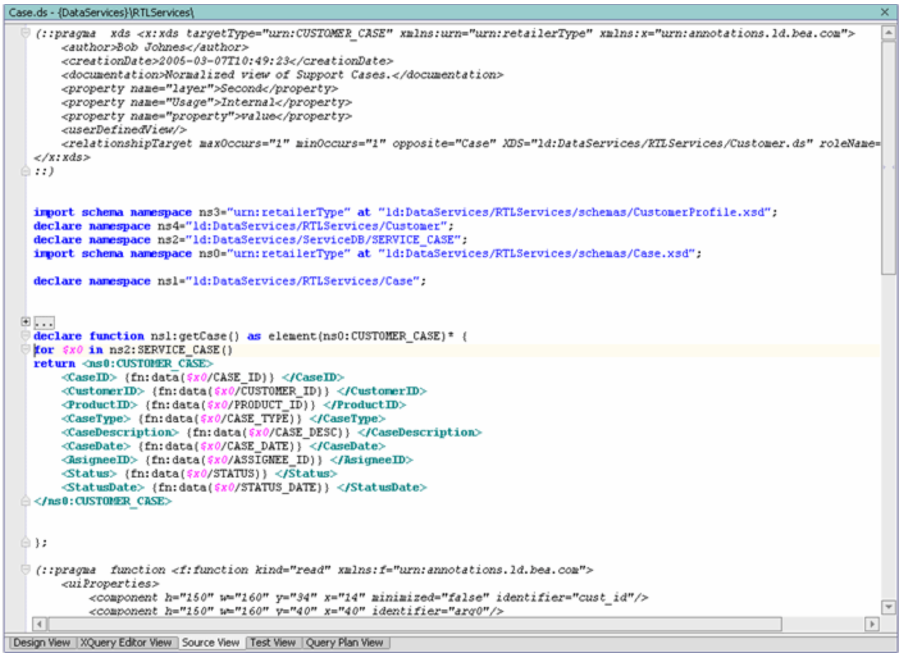
Any work created in XQuery Editor View is immediately reflected in Source View, which permits you to augment the graphical approach to constructing queries with direct work on the XQuery syntax. Two-way editing is supported. Changes you make in Source View are reflected in XQuery Editor View, and vice versa.

Source View

Source View lets you view and/or modify a data service's XQuery annotated source code. Although ALDSP provides extensive visual design tools for developing a data service, sometimes you may need to work directly with the underlying XQuery syntax.

Two-way editing is supported. Changes you make in Source View are reflected in XQuery Editor View, and vice versa.

Figure 1-9 Source View



```
(::pragma  xds <x:xds targetType="urn:CUSTOMER_CASE" xmlns:urn="urn:retailerType" xmlns:x="urn:annotations.ld.bea.com">
    <author>Bob Johnes</author>
    <creationDate>2005-03-07T10:49:23</creationDate>
    <documentation>Normalized view of Support Cases.</documentation>
    <property name="layer">Second</property>
    <property name="Usage">Internal</property>
    <property name="property">value</property>
    <userDefinedView/>
    <relationshipTarget maxOccurs="1" minOccurs="1" opposite="Case" XDS="ld:DataServices/RTLServices/Customer.ds" roleName=
</x:xds>
::)


import schema namespace ns3="urn:retailerType" at "ld:DataServices/RTLServices/schemas/CustomerProfile.xsd";
declare namespace ns4="ld:DataServices/RTLServices/Customer";
declare namespace ns2="ld:DataServices/ServiceDB/SERVICE_CASE";
import schema namespace ns0="urn:retailerType" at "ld:DataServices/RTLServices/schemas/Case.xsd";

declare namespace ns1="ld:DataServices/RTLServices/Case";


...
declare function ns1:getCase() as element(ns0:CUSTOMER_CASE)* {
for $x0 in ns2:SERVICE_CASE()
return <ns0:CUSTOMER_CASE>
    <CaseID> {fn:data($x0/CASE_ID)} </CaseID>
    <CustomerID> {fn:data($x0/CUSTOMER_ID)} </CustomerID>
    <ProductID> {fn:data($x0/PRODUCT_ID)} </ProductID>
    <CaseType> {fn:data($x0/CASE_TYPE)} </CaseType>
    <CaseDescription> {fn:data($x0/CASE_DESC)} </CaseDescription>
    <CaseDate> {fn:data($x0/CASE_DATE)} </CaseDate>
    <AsigneeID> {fn:data($x0/ASSIGNEE_ID)} </AsigneeID>
    <Status> {fn:data($x0/STATUS)} </Status>
    <StatusDate> {fn:data($x0/STATUS_DATE)} </StatusDate>
</ns0:CUSTOMER_CASE>


};

(::pragma  function <f:function kind="read" xmlns:f="urn:annotations.ld.bea.com">
    <uiProperties>
        <component h="150" w="160" y="34" x="14" minimized="false" identifier="cust_id"/>
        <component h="150" w="160" y="40" x="40" identifier="arg0"/>
```

If Source View is not open, complete the following steps:

1. Open a data service such as Case.ds located in DataServices/RTLServices.

2. Select the Source View tab.

Within Source View, you can use the XQuery Construct Palette, which lets you add any of several built-in generic FLWOR statements to the XQuery syntax. You can then customize the generic statement to match your particular needs.

To add a FLWOR construct, drag and drop the selected item into the appropriate declare function space.

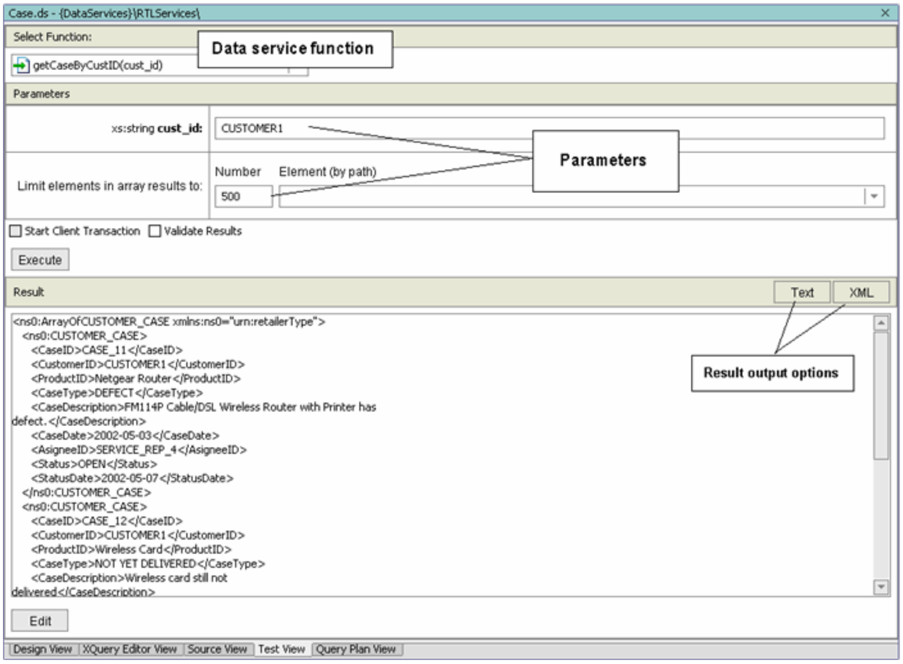
If XQuery Construct Palette is not open, choose View → Windows → XQuery Construct Palette.

Test View

Test View provides a means of running developed query functions within the IDE. Options available in Test View depend on the query being tested. For example, if the query supports parameters, then the Parameters section appears, providing a field for each parameter required by the query.

Using Test View, you can select a specific function, specify appropriate parameters, and execute the query to determine that it is functioning properly. In addition, you can edit the results of the query and pass the modifications back to the underlying data source.

Figure 1-10 Test View



```
<ns0:ArrayOfCUSTOMER_CASE xmlns:ns0="urn:retailerType">
  <ns0:CUSTOMER_CASE>
    <CaseID>CASE_11</CaseID>
    <CustomerID>CUSTOMER1</CustomerID>
    <ProductID>Netgear Router</ProductID>
    <CaseType>DEFECT</CaseType>
    <CaseDescription>FM114P Cable/DSL Wireless Router with Printer has
defect.</CaseDescription>
    <CaseDate>2002-05-03</CaseDate>
    <AsigneeID>SERVICE_REP_4</AsigneeID>
    <Status>OPEN</Status>
    <StatusDate>2002-05-07</StatusDate>
  </ns0:CUSTOMER_CASE>
  <ns0:CUSTOMER_CASE>
    <CaseID>CASE_12</CaseID>
    <CustomerID>CUSTOMER1</CustomerID>
    <ProductID>Wireless Card</ProductID>
    <CaseType>NOT YET DELIVERED</CaseType>
    <CaseDescription>Wireless card still not
delivered</CaseDescription>
```

If Test View is not open, complete the following steps:

1. Open a data service such as `Case.ds` located in DataServices/RTLServices.
2. Select the Test View tab.

1.3 Starting WebLogic Server

WebLogic Server need not be running while you are designing a ALDSP project. However, before you import source metadata or test a developed function, you must start an instance of WebLogic Server.

Any ALDSP projects that you create will run on your system's installation of WebLogic Server, at least until you deploy them.

Note: Multiple versions of WebLogic Server can exist, even on local, sample systems. If you have previously run an instance of WebLogic Server you should shut down that server and change your WebLogic Workshop server settings. This can be done through the Workshop ToolsApplication Properties dialog box.

Objectives

In this exercise, you will:

- Discover ways to start WebLogic Server.
- Confirm that your server is running.

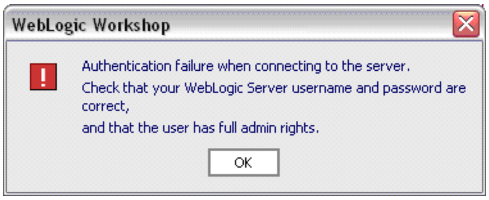
Instructions

There are three ways to start WebLogic Server. Start the server using one of the following ways:

Menu Command	WebLogic Workshop → Tools → WebLogic Server → Start WebLogic Server
Shortcut Keys	Ctrl + Shift + S
From Status Bar	Right-click the red Server Stopped icon, located at the bottom of the WebLogic Workshop window. Then click Start WebLogic Server.

Starting the WebLogic Server may take some time. During the server startup sequence, you may see the following message box:

Figure 1-11 (Possible) WebLogic Server Startup Message



WebLogic Workshop

Authentication failure when connecting to the server.
Check that your WebLogic Server username and password are correct,
and that the user has full admin rights.

OK

If this box displays, click OK.

When WebLogic Server is running, the WebLogic server icon, which appears on the WebLogic Workshop status bar, will turn green. ● Server Running

1.4 Stopping WebLogic Server

There may be times when you want to stop WebLogic Server while still working within ALDSP for WebLogic Workshop.

Objectives

In this exercise, you will:

- Discover how to stop WebLogic Server.
- Confirm that the server is not running.

Instructions

You can stop WebLogic Server using any one of the following ways:

Menu Command	WebLogic Workshop → Tools → WebLogic Server → Stop WebLogic Server
Shortcut Keys	Ctrl + Shift + T
Procedure	Right-click the green Server Running icon, located at the bottom of the WebLogic Workshop window. Then click Stop WebLogic Server.

Check the WebLogic Server icon of WebLogic Workshop to determine whether WebLogic Server is stopped. If WebLogic Server is stopped, the icon will turn red. ● Server Stopped

1.5 Saving Your Work

As you build your data services, you may want to save your work on a regular basis.

Objectives

In this exercise, you will:

- Discover three ways to save your work while working within the application.
- Discover how to save one or more files when exiting the application or closing WebLogic Workshop.

Instructions

You can save your work using the following commands:

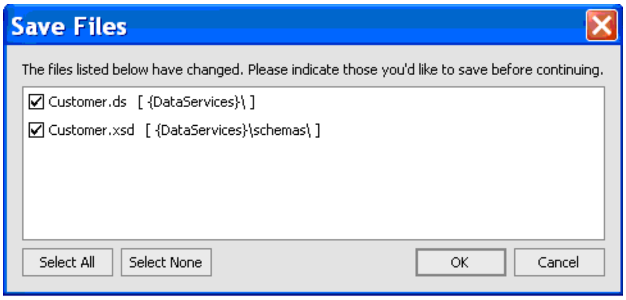
Menu Command	Icon
File → Save	💾

File → Save As	Not Applicable
File → Save All	🖫

Save All is generally recommended for ALDSP applications. The Save As and Save All options are only available if you have made changes to your application.

In addition, if you exit WebLogic Workshop and there are any unsaved changes, you are provided with an option to save either specific or all edited files.

Figure 1-12 Save File Options on Exiting WebLogic Workshop



The files listed below have changed. Please indicate those you'd like to save before continuing.

- [✓] Customer.ds [{DataServices}\ ]
- [✓] Customer.xsd [{DataServices}\schemas\ ]

Lesson Summary

In this lesson, you learned how to:

- Use several of the key tools within ALDSP for WebLogic Workshop environment.
- Start and stop the WebLogic Server.
- Save files within a Data Services application.

Creating a Physical Data Service

A data service is simply a file containing XQuery functions and supporting structured information. The most basic data service is a physical data service, which models a single physical data source residing in a relational database, Web service, flat file, XML file, or Java function.

Data Services Platform approaches the problem of creating integration architectures by building data services around multiple physical data services. Therefore, in this lesson, you will create data services based on relational data included in the sample PointBase database provided with ALDSP:

- Customer Relationship Management (CRM) data, stored in the RTLCUSTOMER database.
- Order Management System (OMS) data for apparel and electronic products, stored in the RTLAPPLOMS and RTLELECOMS databases.
- Customer service data, stored in the RTLSERVICE database.

Objectives

After completing this lesson, you will be able to:

- Create a ALDSP application and project.
- Generate multiple physical data services, based on underlying relational data sources.
- Test a physical data service.

Overview

A data service is a collection of one or several related query functions. The service typically models a unit of enterprise information, such as customer or product data.

The shape of a data service is defined by an XML type that classifies each data element as a particular form of information, according to its allowable contents and units of data. For example, an xs:string type can be a sequence of alphabetic, numeric, and/or special characters, while an xs:date type can only be numeric characters presented in a YYYY-MM-DD format. ALDSP uses the XML type to model and normalize disparate data into a unified view.

The data service interface consists of public functions that enable client-based consuming applications to retrieve data from the modeled data source.

2.1 Creating an ALDSP Application

Because a data service is part of a specific ALDSP project, and a project is part of a single WebLogic Workshop application, you will first need to create the application, and then a project, before creating a physical data service. (Alternatively, an existing application could be used; in that case you would simply create a ALDSP project within the application.)

An application, which is deployed as a single unit to an instance of WebLogic Server, is a J2EE enterprise application that ultimately produces a J2EE Enterprise Application Archive (EAR) file. This, in turn, provides you with a multi-user application that is ready for Internet deployment. Except in specific cases, such as accessing remote EJBs or Web services, an application is self-contained. The application's components may reference each other, but may not generally reference components in other applications. An application's components include:

- One or more projects, data services, schemas, and libraries.
- Zero or more modules and security roles.

An application should represent a related collection of business solutions. For example, if you are deploying two Web sites — one an e-commerce site and the other a human resources portal for employees — you would probably create separate WebLogic applications for each.

An application is also the top-level unit of work that you manipulate within the WebLogic Workshop environment. Only one application can be active at a time.

Objectives

In this exercise, you will:

- Create a ALDSP-enabled application.
- Explore default application components.

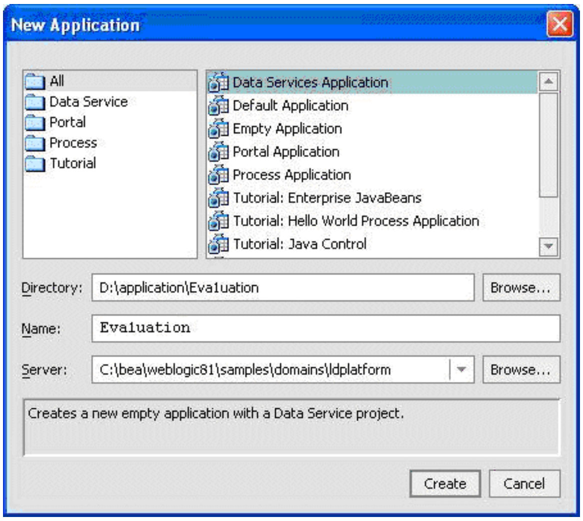
Instructions

1. Choose File → New → Application
2. In the New Application dialog box, select Data Services Application.
3. Enter Evaluation in the Name field.

Note: The sample code used to work on this tutorial uses Evaluation as the application name. Ensure that you name the ALDSP application as Evaluation so that the sample works successfully with your application.

4. Click Create.

Figure 2-1 Creating a ALDSP Application



Directory: D:\application\Evaluation

Name: Evaluation

Server: C:\bea\weblogic81\samples\domains\ldplatform

Creates a new empty application with a Data Service project.

The components of the application are represented in a hierarchical tree structure in the Application pane. When you first create a Data Services application, the following default components are automatically generated:

Data Service project. Takes the name of your application (in this case, Evaluation). Within the project folder, there is initially a single component, the `xquery-types.xsd` file. This file is an XML Schema Definition (XSD) that describes the contents, semantics, and structure of the project.

Modules. Initially an empty folder.

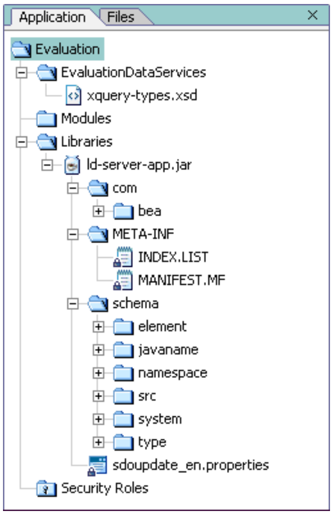
Libraries. Contains the ld-server-app.jar file. This file contains various folders and files, as displayed in Figure 2-2.

Note: Initially, the Libraries folder is empty. The `ld-server-app.jar` file is imported only after you build the Evaluation project.

Security Roles. Initially an empty folder.

Figure 2-2 displays the default folders created for the Evaluation application.

Figure 2-2 Initial Application Structure



2.2 Creating a Data Services Project

A project groups related files—data services, models, and metadata—within an application. Each application can support multiple projects. As you develop the application, you may want to create new projects for the following reasons:

To separate unrelated functionality. Each project should contain closely-related components. For example, if you want to create one or more data services that expose order status to your customers, and also one or more Web services that expose inventory status to your suppliers, you would probably organize these two sets of unrelated Web services into two projects.

To control build units. Each project produces a particular type of file when the project is built. For example, a Java project produces a JAR file. If you want to reuse the Java classes, you would segregate the Java classes into a separate project, and then reference the resulting JAR file from other projects in your application.

Although a default Data Services project is created when you create a new Data Service application, for this tutorial you will create a new project.

Objectives

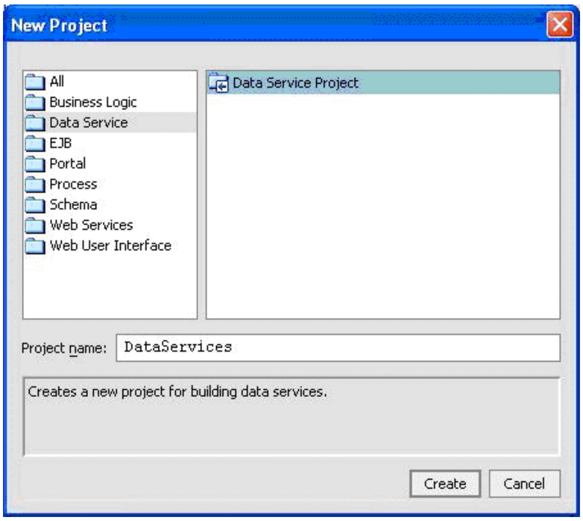
In this exercise, you will:

- Create a new Data Service project.
- Review the results.

Instructions

1. Choose File → New → Project
2. In the New Project dialog box, select Data Service Project.
3. Enter DataServices in the Project name field.
4. Click Create.

Figure 2-3 Creating a New Data Service Project



Project name: DataServices

Creates a new project for building data services.

The components of your new Data Service project are represented in a hierarchical tree structure in the Application pane. At present, there is only one component in the project, the xquery-types .xsd file. This file is an XML schema definition that describes the contents, semantics, and structure of the project.

2.3 Creating Project Sub-Folders

Folders let you logically group different data services, and their associated files, within a single project. For example, if you had three data sources — one relational database containing tables for customer-oriented information and two Web services providing credit rating and information — you would probably want to create two folders, one for the database and one for the Web services.

Objectives

In this exercise, you will:

- Create four sub-folders within the DataServices project folder.
- Review the results.

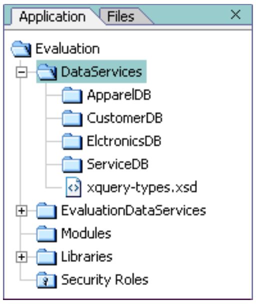
Instructions

1. Right-click the DataServices project folder.
2. Choose New → Folder.
3. Enter CustomerDB in the Name field.
4. Click OK.
5. Repeat steps 1 through 4 to create additional data service folders for:

- ApparelDB
- ElectronicsDB
- ServiceDB

After adding these four folders, your DataServices project folder should look similar to Figure 2-4.

Figure 2-4 Project Sub-Folders



2.4 Importing Relational Source Metadata

When you installed DSP, several sample data sources were also installed. One such sample data source is the Avitek RTL PointBase database. It contains a number of relational database schemas that provide the metadata needed to build your physical data services, including:

- Customer Relationship Management (CRM) data, stored in the RTLCUSTOMER database.
- Order Management System (OMS) data for apparel products, stored in the RTLAPPLOMS database.

- Order Management System (OMS) data electronic products, stored in the RTLELECOMS database.
- Customer service data, stored in the RTLSERVICE database.

A physical data service, which models physical data existing somewhere in your enterprise, is automatically generated when you import relational source metadata. Each generated physical data service represents a single data source that can be integrated with other physical or logical data services.

Objectives

In this exercise, you will:

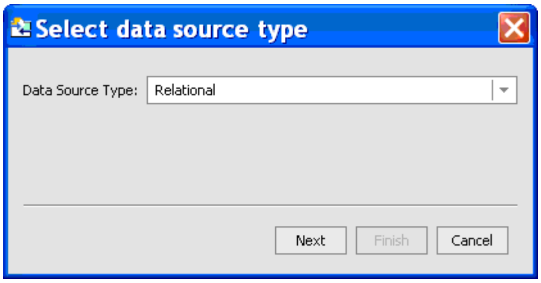
- Import source metadata from four RTL PointBase databases, thereby generating multiple physical data services.
- Review the results.

Instructions

Note: WebLogic Server must be running. If it is not already running, start the server (see 1.3 Starting WebLogic Server) before you begin this exercise.

1. Right-click the CustomerDB folder.
2. Choose Import Source Metadata from the pop-up menu.
3. Select Relational from the Data Source Type drop-down list and click Next.

Figure 2-5 Select Data Source


Select data source type

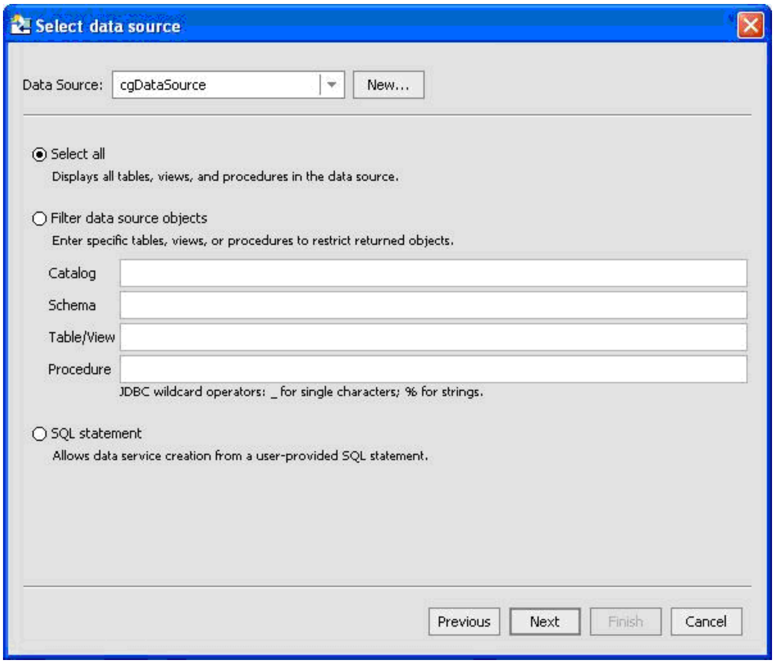
Data Source Type: Relational

4. Specify the data source, by completing the following steps:

a. Select cgDataSource from the Data Source drop-down list.

b. Click Select All and then click Next.

Figure 2-6 Select Data Source



Select data source

Data Source: cgDataSource [New...]

(●) Select all
Displays all tables, views, and procedures in the data source.

( ) Filter data source objects
Enter specific tables, views, or procedures to restrict returned objects.

Catalog	
Schema	
Table/View	
Procedure	

JDBC wildcard operators: _ for single characters; % for strings.

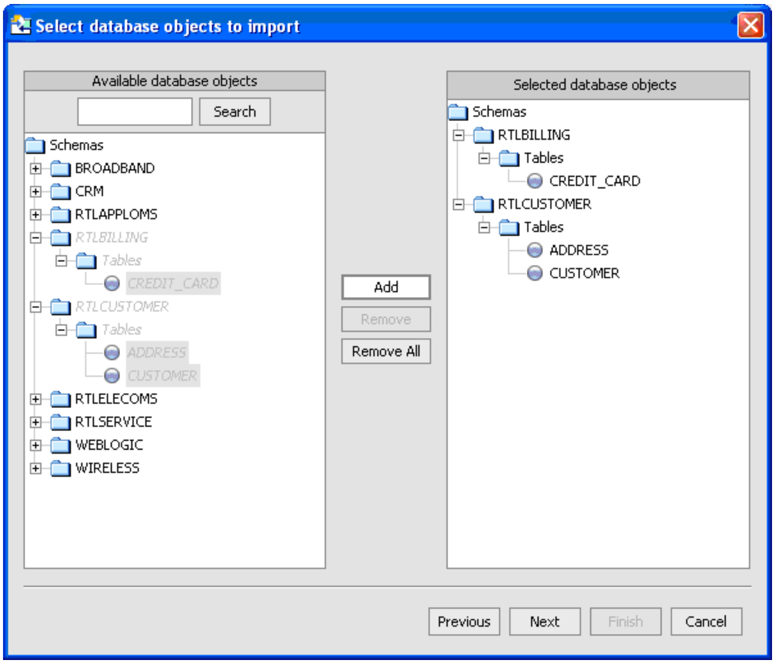
( ) SQL statement
Allows data service creation from a user-provided SQL statement.

WebLogic Server fetches the specified data, and then displays the Select Database Objects to Import dialog box. The source metadata for each selected object will be used to generate a physical data service.

5. Expand the RTLCUSTOMER and RTLBILLING folders, located in the left pane.

6. Select all tables from both schemas and click Add. The selected objects display in the right pane.

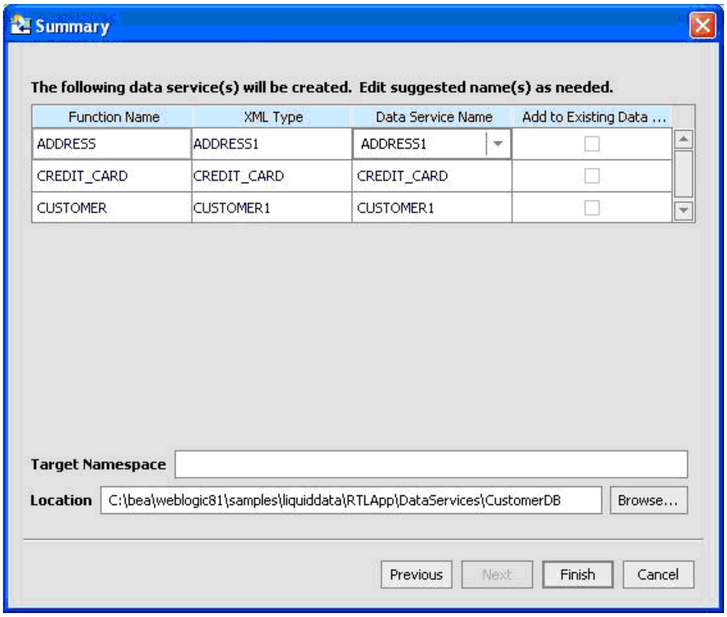
Figure 2-7 Selected Database Objects to Import



7. Click Next. A Summary dialog box opens, displaying the following information:

- XML type, for database objects whose source metadata will be imported.
- Data Service Name, for each data service that will be generated from the source metadata. (Any name conflicts appear in red; you can modify any data service name.)
- Target Namespace, for the data service being generated. This is optional.
- Location, where the generated data services will reside.

Figure 2-8 Summary



The following data service(s) will be created. Edit suggested name(s) as needed.

Function Name	XML Type	Data Service Name	Add to Existing Data ...
ADDRESS	ADDRESS1	ADDRESS1	☐
CREDIT_CARD	CREDIT_CARD	CREDIT_CARD	☐
CUSTOMER	CUSTOMER1	CUSTOMER1	☐

Target Namespace

Location C:\bea\weblogic81\samples\liquiddata\RTLApp\DataServices\CustomerDB

8. Click Finish.

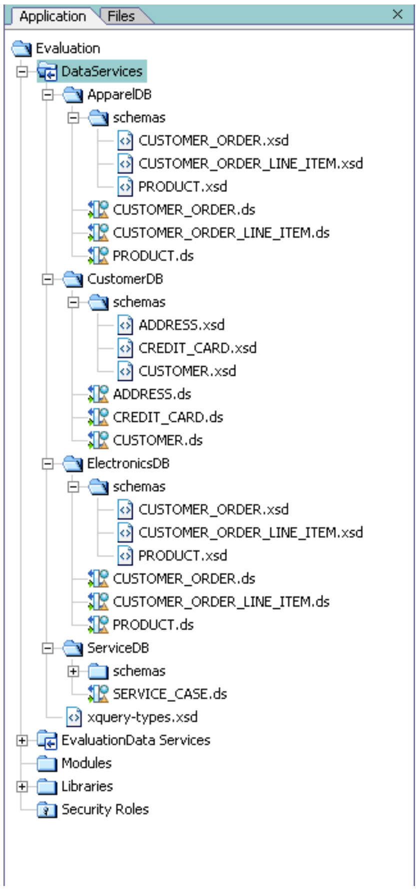
9. Repeat steps 1 through 8 to import source metadata into the ApparelDB, ElectronicsDB, and ServiceDB folders, substituting the following information for steps 1 and 5:

Table 2-9 Data Service Objects and Data Source

Data Service Objects	Data Source
ApparelDB	RTLAPPLOMS
ElectronicsDB	RTLELECOMS
ServiceDB	RTLSERVICE

The Application pane should appear similar to Figure 2-10. If you expand a data service's schema folder, you will see XSD files for each data service generated from the underlying data source.

Figure 2-10 New Data Services



2.5 Building a Project

Building a project simply means that the project's source code is compiled into machine-readable instructions. Each project produces a particular type of file when the project is built. For example, a Java project produces a JAR file.

Objectives

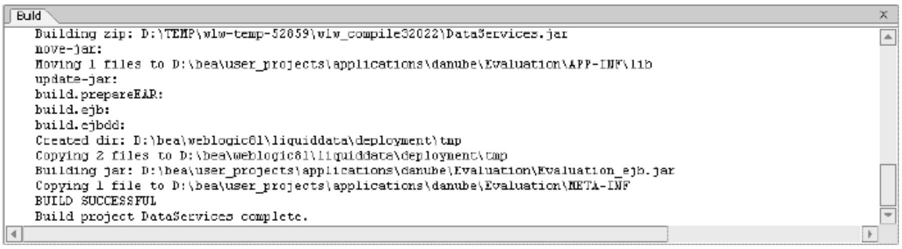
In this exercise, you will:

- Build the DataServices project.
- Review the results in the Build window.

Instructions

1. Right-click the DataServices project folder.
2. Choose Build DataServices. It may take a few moments for the project to be built. When complete, you will see a message in the Build window, similar to that displayed in Figure 2-11. (If the Build window is not open, choose View → Windows → Build or press Alt+5.)

Figure 2-11 Build Project Information



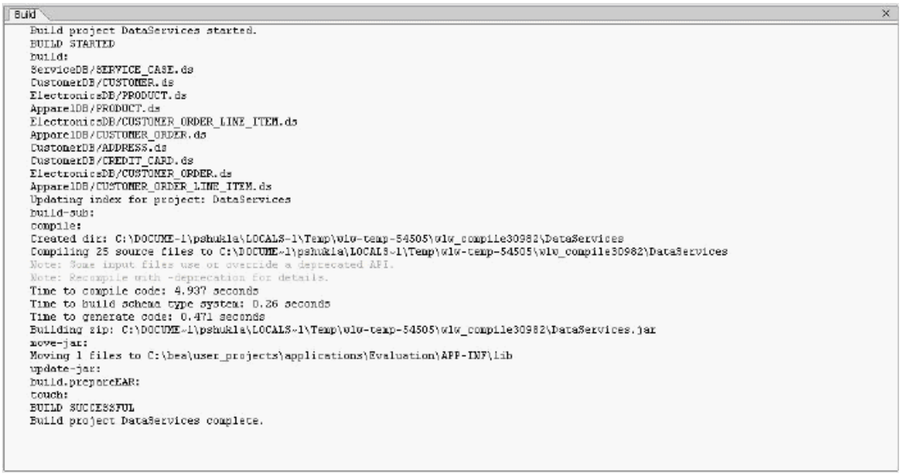
```
Building zip: D:\TEMP\wlw-temp-52859\wlw_compile32022\DataServices.jar
move-jar:
Moving 1 files to D:\bea\user_projects\applications\danube\Evaluation\APP-INF\lib
update-jar:
build.prepareEAR:
build.ejb:
build.ejbdd:
Created dir: D:\bea\weblogic81\liquiddata\deployment\tmp
Copying 2 files to D:\bea\weblogic81\liquiddata\deployment\tmp
Building jar: D:\bea\user_projects\applications\danube\Evaluation\Evaluation_ejb.jar
Copying 1 file to D:\bea\user_projects\applications\danube\Evaluation\META-INF
BUILD SUCCESSFUL
Build project DataServices complete.
```

3. Scroll through the Build window. As part of the Build process, ALDSP generates a number of files, including the following:

- Data service (`.ds`) files for each table within the underlying data source.
- Miscellaneous JAR and EJB files.

Figure 2-12 displays the complete Build information for the DataServices project.

Figure 2-12 Complete Build Information for the DataServices Project



```
Build project DataServices started.
BUILD STARTED
build:
ServiceDB/SERVICE_CASE.ds
CustomerDB/CUSTOMER.ds
ElectronicsDB/PRODUCT.ds
ApparelDB/PRODUCT.ds
ElectronicsDB/CUSTOMER_ORDER_LINE_ITEM.ds
ApparelDB/CUSTOMER_ORDER.ds
CustomerDB/ADDRESS.ds
CustomerDB/CREDIT_CARD.ds
ElectronicsDB/CUSTOMER_ORDER.ds
ApparelDB/CUSTOMER_ORDER_LINE_ITEM.ds
Updating index for project: DataServices
build-sub:
compile:
Created dir: C:\DOCUME~1\pshukla\LOCALS~1\Temp\wlw-temp-54505\wlw_compile30982\DataServices
Compiling 25 source files to C:\DOCUME~1\pshukla\LOCALS~1\Temp\wlw-temp-54505\wlw_compile30982\DataServices
Note: Some input files use or override a deprecated API.
Note: Recompile with -deprecation for details.
Time to compile code: 4.937 seconds
Time to build schema type system: 0.26 seconds
Time to generate code: 0.471 seconds
Building zip: C:\DOCUME~1\pshukla\LOCALS~1\Temp\wlw-temp-54505\wlw_compile30982\DataServices.jar
move-jar:
Moving 1 files to C:\bea\user_projects\applications\Evaluation\APP-INF\lib
update-jar:
build.prepareEAR:
touch:
BUILD SUCCESSFUL
Build project DataServices complete.
```

4. *(Optional)* Expand the Libraries folder. You should see the `DataServices.jar` file.

2.6 Viewing Physical Data Service Information

A physical data service is automatically generated when you import source metadata and build the associated project. Each generated physical data service represents a single data source that can be integrated with other physical or logical data services.

When ALDSP generates a physical data service, it also generates XML data types, an XML Schema Definition (`.xsd` file), default query and navigation functions, and pragma information.

Objectives

In this exercise, you will:

- View XML type, native data types, XML schema definition, generated functions, and metadata.
- Use Design View and Source View to obtain information about a data service.

Viewing XML type

An XML type, which derives from the data service's XML Schema Definition (XSD), is a structured XML document that classifies each element within the data service as a particular form of information, according to its allowable contents and units of data. For example, the XML type for the CUSTOMER data service is CUSTOMER, whose elements include:

- CUSTOMER_ID, whose xs:string classification indicates the element's return data will be formatted as a sequence of alphabetic, numeric, and/or special characters.
- CUSTOMER_SINCE, whose xs:date classification indicates the element's return data will be formatted as numeric characters presented in a YYYY-MM-DD format.

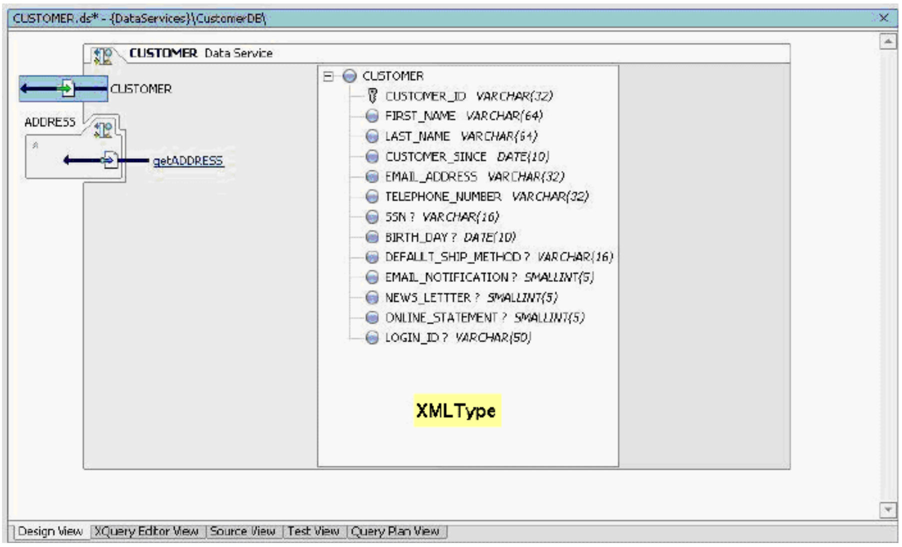
Multiple data services can use a single XML type. ALDSP uses the XML type as the default superset of data elements that will be returned by a set of queries. This superset XML type, known as the Return type, models and normalizes data retrieved from the underlying data source, thereby transforming disparate data into a unified view.

Instructions

1. In the Application pane, expand the CustomerDB folder.
2. Double-click the `CUSTOMER.ds` file. The data service opens in Design View.

   Note: The data service automatically opens in the View workspace last used; if Design View is not currently open, click the Design View tab.
3. In the middle of the data service representation you should see the `CUSTOMER XML` (also known as schemas) type for the data service, plus the XML classification for each element in the data service. Items marked with a question mark (?) are optional elements, which indicates: 1) if there is no data in the underlying data source, that element will not display in the data set returned by the data service and 2) a query function can succeed without providing any value for that particular element.

Figure 2-13 Design View of XML Type



Viewing Native Data Type

A Native Data Type classifies each data element according to the definitions specified in the underlying data source. For relational data sources, ALDSP generates Native Data Type definitions based on the underlying database's table structure and column data definitions.

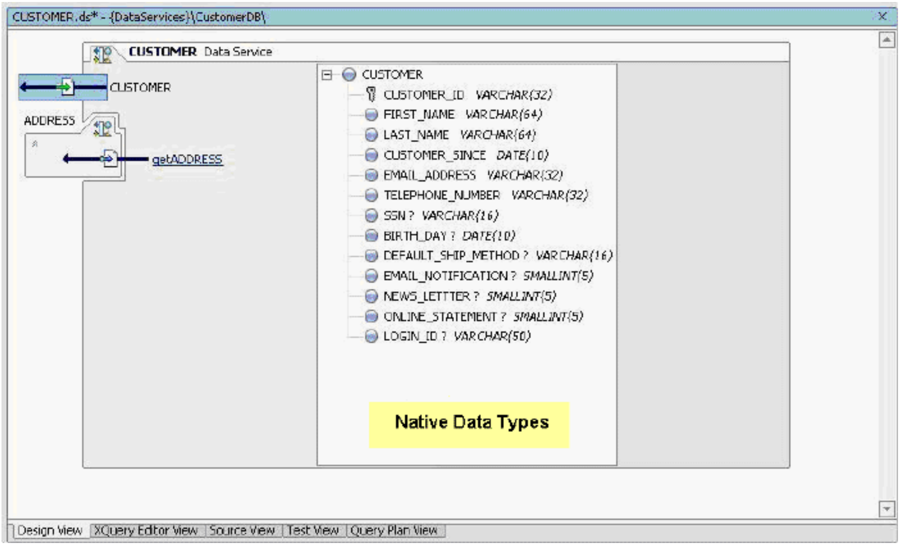
Instructions

1. Right-click the CUSTOMER Data Service header on the Design View tab. (You can also right-click any empty space within the data service diagram.)

2. Select Display Native Type. This will display the original data type for each element in the underlying data source.

3. In the middle of the data service representation, you should see Native Types for each data element in the data service.

Figure 2-14 Design View of Native Type



```
CUSTOMER
    CUSTOMER_ID  VARCHAR(32)
    FIRST_NAME  VARCHAR(64)
    LAST_NAME  VARCHAR(64)
    CUSTOMER_SINCE  DATE(10)
    EMAIL_ADDRESS  VARCHAR(32)
    TELEPHONE_NUMBER  VARCHAR(32)
    SSN ?  VARCHAR(16)
    BIRTH_DAY ?  DATE(10)
    DEFAULT_SHIP_METHOD ?  VARCHAR(16)
    EMAIL_NOTIFICATION ?  SMALLINT(5)
    NEWS_LETTTER ?  SMALLINT(5)
    ONLINE_STATEMENT ?  SMALLINT(5)
    LOGIN_ID ?  VARCHAR(50)
```

Viewing XML Schema Definition

An XML Schema Definition file (`.xsd`) corresponds exactly to the XML type of a data service. It defines the structure and content of an XML document, such as the XML type document. In other words, it defines the vocabulary, rules, and conventions for representing information in a system.

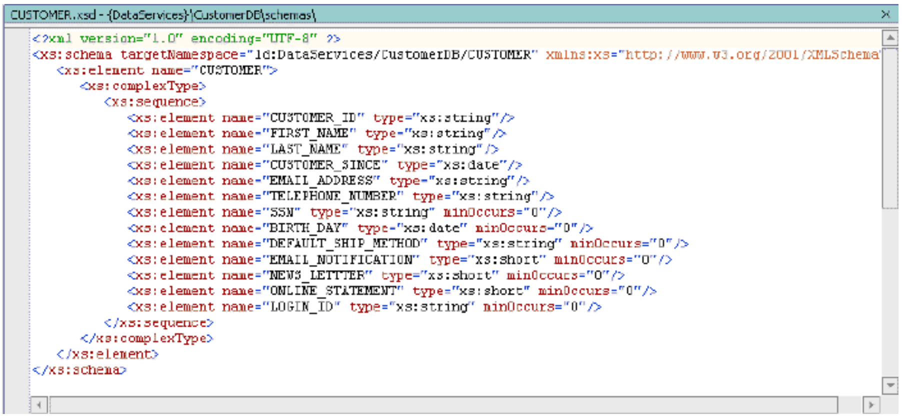
An .xsd file is organized as a flat catalog of complex elements, any attributes, and any child elements. For physical data services, ALDSP automatically generates a .xsd file from underlying data when the underlying data source's metadata is imported. Generated .xsd files are placed in the appropriate data service's schema directory.

Note: For logical data services, you must create a schema. You can use XQuery Editor View, discussed in Tutorial 3, "Creating a Logical Data Service", to create such schemas (XSD files).

Instructions

1. Right-click the CUSTOMER element, located in the XML type pane. A pop-up menu opens.
2. Choose Go to Source to view the underlying schema information.

Figure 2-15 XML Schema Definition



CUSTOMER.xsd - {DataServices}\CustomerDB\schemas\

```
<?xml version="1.0" encoding="UTF-8" ?>
<xs:schema targetNamespace="ld:DataServices/CustomerDB/CUSTOMER" xmlns:xs="http://www.w3.org/2001/XMLSchema"
   <xs:element name="CUSTOMER">
      <xs:complexType>
         <xs:sequence>
            <xs:element name="CUSTOMER_ID" type="xs:string"/>
            <xs:element name="FIRST_NAME" type="xs:string"/>
            <xs:element name="LAST_NAME" type="xs:string"/>
            <xs:element name="CUSTOMER_SINCE" type="xs:date"/>
            <xs:element name="EMAIL_ADDRESS" type="xs:string"/>
            <xs:element name="TELEPHONE_NUMBER" type="xs:string"/>
            <xs:element name="SSN" type="xs:string" minOccurs="0"/>
            <xs:element name="BIRTH_DAY" type="xs:date" minOccurs="0"/>
            <xs:element name="DEFAULT_SHIP_METHOD" type="xs:string" minOccurs="0"/>
            <xs:element name="EMAIL_NOTIFICATION" type="xs:short" minOccurs="0"/>
            <xs:element name="NEWS_LETTTER" type="xs:short" minOccurs="0"/>
            <xs:element name="ONLINE_STATEMENT" type="xs:short" minOccurs="0"/>
            <xs:element name="LOGIN_ID" type="xs:string" minOccurs="0"/>
         </xs:sequence>
      </xs:complexType>
   </xs:element>
</xs:schema>
```

3. After reviewing the XSD, click the Close box (X) in the upper-right corner of the source pane to return to Design View of your data service.

Note: Clicking the large red X will close WebLogic Workshop.

Viewing Generated Functions

The data service interface consists of public functions of the data service, which can be of several types:

- One or more read functions, which typically return data in the form of the data service XML type.
- One or more navigation functions, which return data from related data services. The navigation functions are based on any relationships defined within the underlying data source. Relationships enhance the flexibility of data services by enabling the return of data in the shape of another data service.
- One submit() function, which allows users to persist changes to the original data source. The submit() function does not appear in Design View.

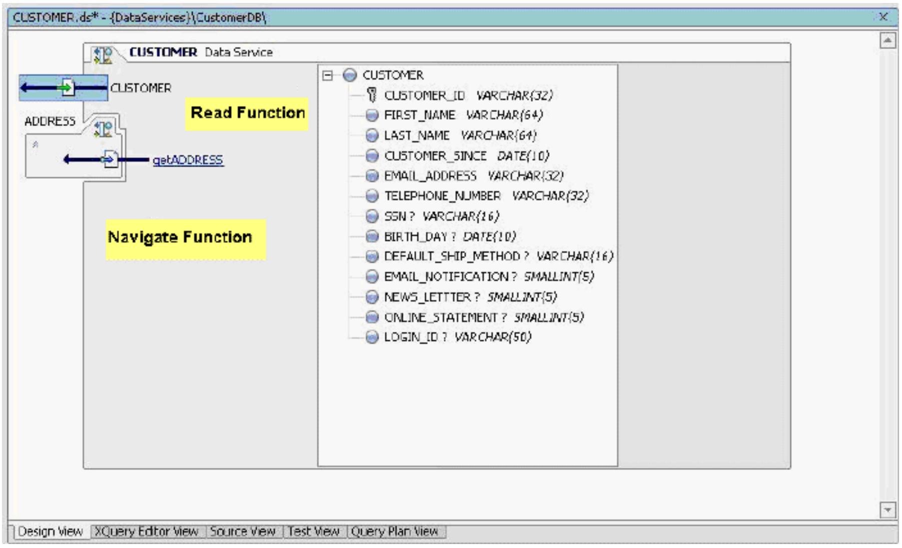
In addition to public functions, a data service can include private functions and side effect functions. Private functions are only used within the data service. They generally contain common processing logic that can be used by more than one data service function. Side effect functions can be invoked from the client side. For example, a side effect function can contain code to update a non-RDBMS data source, such as xml, flat files, and Web services, and clients can invoke this function to perform updates. (For more information, see the *Data Service Developer's Guide*.)

Instructions

1. In Design View, notice the public functions displayed in the left pane of the diagram. These functions, which were generated for the data service, include the following:

- `CUSTOMER()`, a read function that retrieves data from the underlying RTLCUSTOMER database.
- `getADDRESS()`, a navigate function that retrieves data from the ADDRESS data service. This function is based on a relationship between the CUSTOMER and ADDRESS tables, which are defined in the RTLCUSTOMER database.

Figure 2-16 Design View: Generated Functions



```
CUSTOMER
  CUSTOMER_ID  VARCHAR(32)
  FIRST_NAME  VARCHAR(64)
  LAST_NAME  VARCHAR(64)
  CUSTOMER_SINCE  DATE(10)
  EMAIL_ADDRESS  VARCHAR(32)
  TELEPHONE_NUMBER  VARCHAR(32)
  SSN ?  VARCHAR(16)
  BIRTH_DAY ?  DATE(10)
  DEFAULT_SHIP_METHOD ?  VARCHAR(16)
  EMAIL_NOTIFICATION ?  SMALLINT(5)
  NEWS_LETTTER ?  SMALLINT(5)
  ONLINE_STATEMENT ?  SMALLINT(5)
  LOGIN_ID ?  VARCHAR(50)
```

2. (Optional) Right-click the CUSTOMER Data Service header and choose Display XML type from the pop-up menu. (You can also right-click any empty space within the data service diagram.)

Viewing Data Service Metadata

Metadata is simply information about the structure of data; it provides facts about the data service's data, format, meaning, and lineage. For example, a list of tables and columns within a database is metadata. ALDSP uses metadata to describe a data service: what information is provided by the data service and the information's lineage (that is, the source for the information.)

In addition to documenting data services for potential consumers, metadata helps you determine what data services are affected when inevitable changes occur in the underlying data source layer. Of course in the case of physical data services, the metadata primarily describes metadata extracted from the physical data source.

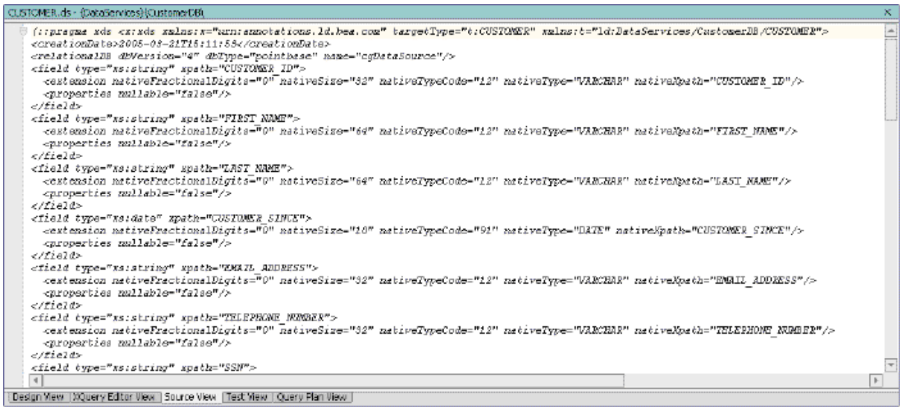
Metadata information is contained in the data service's META-INF folder. Normally you should not need to refer to the contents of this folder.

Instructions

1. Select the Source View tab. The metadata information used by the Customer data service appears. (Also available in Source View are data service namespace, schema namespace, and XQuery functions.)
2. Click the + icon to display all metadata information.
3. Notice the following:

- The date the data service was created.
- The data source from which the metadata was imported.
- The XML type, XPath, Native Data Type, and native XPath for each element within the data service.
- The relationship target, role name, role number, XDS, and relationship parameters for each data service associated with the active data service.

Figure 2-17 Source View of Metadata



```
(::pragma xds <x:xds xmlns:x="urn:annotations.ld.bea.com" targetType="t:CUSTOMER" xmlns:t="ld:DataServices/CustomerDB/CUSTOMER">
<creationDate>2005-08-21T15:11:55</creationDate>
<relationalDB dbVersion="4" dbType="pointbase" name="cgDataSource"/>
<field type="xs:string" xpath="CUSTOMER_ID">
  <extension nativeFractionalDigits="0" nativeSize="32" nativeTypeCode="12" nativeType="VARCHAR" nativeXpath="CUSTOMER_ID"/>
  <properties nullable="false"/>
</field>
<field type="xs:string" xpath="FIRST_NAME">
  <extension nativeFractionalDigits="0" nativeSize="64" nativeTypeCode="12" nativeType="VARCHAR" nativeXpath="FIRST_NAME"/>
  <properties nullable="false"/>
</field>
<field type="xs:string" xpath="LAST_NAME">
  <extension nativeFractionalDigits="0" nativeSize="64" nativeTypeCode="12" nativeType="VARCHAR" nativeXpath="LAST_NAME"/>
  <properties nullable="false"/>
</field>
<field type="xs:date" xpath="CUSTOMER_SINCE">
  <extension nativeFractionalDigits="0" nativeSize="10" nativeTypeCode="91" nativeType="DATE" nativeXpath="CUSTOMER_SINCE"/>
  <properties nullable="false"/>
</field>
<field type="xs:string" xpath="EMAIL_ADDRESS">
  <extension nativeFractionalDigits="0" nativeSize="32" nativeTypeCode="12" nativeType="VARCHAR" nativeXpath="EMAIL_ADDRESS"/>
  <properties nullable="false"/>
</field>
<field type="xs:string" xpath="TELEPHONE_NUMBER">
  <extension nativeFractionalDigits="0" nativeSize="32" nativeTypeCode="12" nativeType="VARCHAR" nativeXpath="TELEPHONE_NUMBER"/>
  <properties nullable="false"/>
</field>
<field type="xs:string" xpath="SSN">
```

Note: Before you test any function or data service, you should ideally clean and redeploy the application, so that the data is updated on the WebLogic server also.

4. To clean the application, right-click Evaluation and select Clean Application.

5. To redeploy the application, right-click Evaluation and select Deployment → Redeploy.

2.7 Testing Physical Data Service Functions

Testing a data service's functionality within Test View lets you determine whether the data service is able to return the expected data results.

Objectives

In this exercise, you will:

- Test the `CUSTOMER()` function.
- Review the results in Test View.
- Review the results in the Output window to confirm that the data is pulled from the correct data source.

Instructions

1. Select the Test View tab.

2. Select `CUSTOMER()` from the function drop-down list.

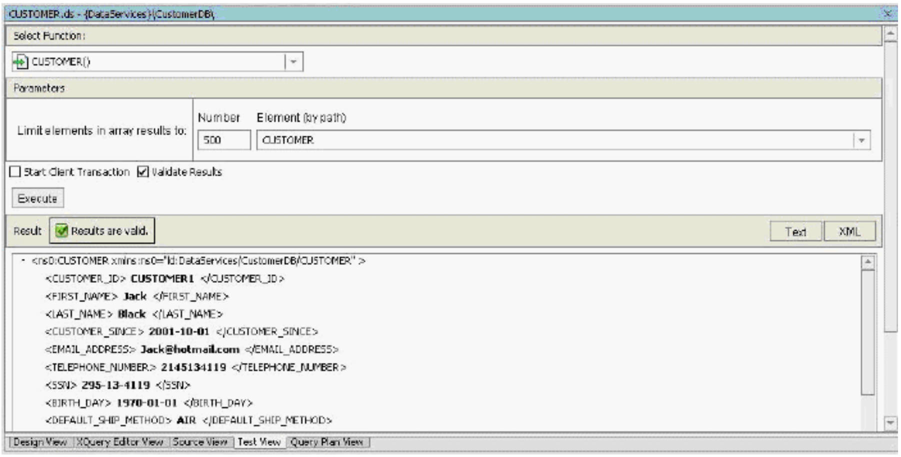
3. Click Execute. You should see data returned from the RTLCUSTOMER database, formatted according to the CUSTOMER data service's Return type, which is defined by each element's XML type.

Note: At times the WebLogic server may not get updated automatically. In that case, you may get some validation errors when you execute the function. To fix this, try cleaning and redeploying the application.

4. Expand the nodes and notice the following:

   Each element defined by the XML type returns specific data retrieved from the RTLCUSTOMER database. For example, the <FIRST_NAME> element returns "Jack" as an xs:string, while the <CUSTOMER_SINCE> element returns "2001-10-01" as an xs:date.

Figure 2-18 Physical Data Service Test Results



CUSTOMER()

Limit elements in array results to: Number 500 Element (by path) CUSTOMER

☐ Start Client Transaction ☑ Validate Results

Results are valid.

```
- <ns0:CUSTOMER xmlns:ns0="ld:DataServices/CustomerDB/CUSTOMER" >
    <CUSTOMER_ID> CUSTOMER1 </CUSTOMER_ID>
    <FIRST_NAME> Jack </FIRST_NAME>
    <LAST_NAME> Black </LAST_NAME>
    <CUSTOMER_SINCE> 2001-10-01 </CUSTOMER_SINCE>
    <EMAIL_ADDRESS> Jack@hotmail.com </EMAIL_ADDRESS>
    <TELEPHONE_NUMBER> 2145134119 </TELEPHONE_NUMBER>
    <SSN> 295-13-4119 </SSN>
    <BIRTH_DAY> 1970-01-01 </BIRTH_DAY>
    <DEFAULT_SHIP_METHOD> AIR </DEFAULT_SHIP_METHOD>
```

5. To view the results in the Output window, you need to enable auditing in the ALDSP console. To enable auditing:

   a. Open the ALDSP console, typically located at `http://localhost:7001/ldconsole`.

   b. Log on using the following credentials:

      - User = weblogic
      - Password = weblogic

c. Expand ldplatform in the left-hand menu and click Evaluation.

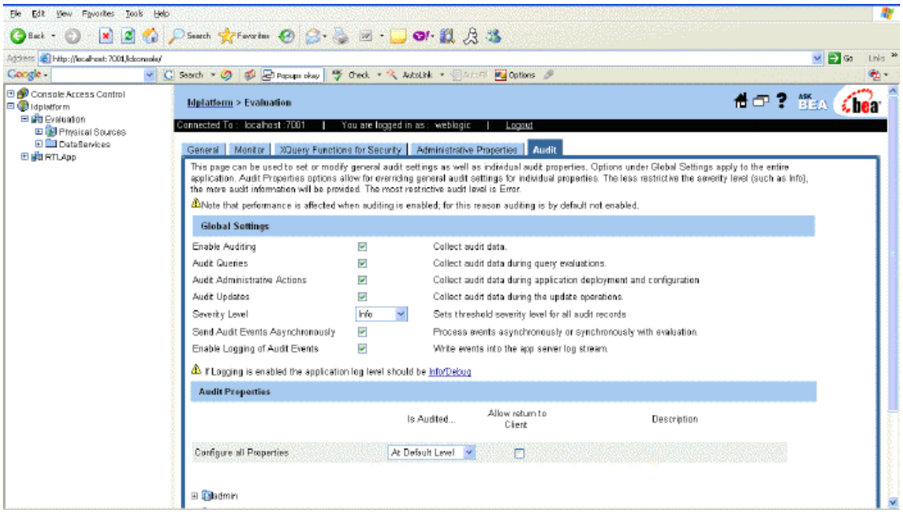
d. Click the Audit tab.

e. Select the following options in the Global Settings section: Enable Auditing, Audit Queries, Audit Administrative Actions, Audit Updates, Send Audit Events Asynchronously, and Enable Logging of Audit Events (Figure 2-19).

f. Select the At Default Level option from the Configure all Properties list in Audit Properties.

g. Click Apply.

Figure 2-19 Audit Tab in the ALDSP Console



ldplatform > Evaluation

Connected To : localhost :7001 | You are logged in as : weblogic | Logout

This page can be used to set or modify general audit settings as well as individual audit properties. Options under Global Settings apply to the entire application. Audit Properties options allow for overriding general audit settings for individual properties. The less restrictive the severity level (such as Info), the more audit information will be provided. The most restrictive audit level is Error.

Note that performance is affected when auditing is enabled; for this reason auditing is by default not enabled.

Global Settings

Enable Auditing	✓	Collect audit data.
Audit Queries	✓	Collect audit data during query evaluations.
Audit Administrative Actions	✓	Collect audit data during application deployment and configuration
Audit Updates	✓	Collect audit data during the update operations.
Severity Level	Info	Sets threshold severity level for all audit records
Send Audit Events Asynchronously	✓	Process events asynchronously or synchronously with evaluation.
Enable Logging of Audit Events	✓	Write events into the app server log stream.

If Logging is enabled the application log level should be Info/Debug

Audit Properties

	Is Audited...	Allow return to Client	Description
Configure all Properties	At Default Level	☐	

admin

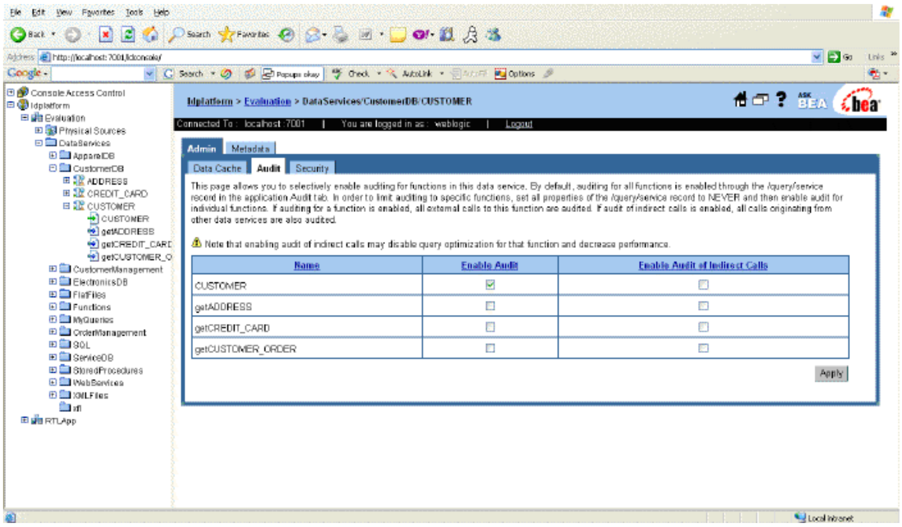
h. In the left-hand menu, expand Evaluation, DataServices, and then CustomerDB as shown in Figure 2-20.

i. Click Customer and select the Admin tab.

j. Click the Audit tab.

k. Select the check box in the Enable Audit column for the CUSTOMER function.

Figure 2-20 Enabling Function-Level Auditing


ldplatform > Evaluation > DataServices/CustomerDB/CUSTOMER

Connected To : localhost :7001 | You are logged in as : weblogic | Logout

This page allows you to selectively enable auditing for functions in this data service. By default, auditing for all functions is enabled through the /query/service record in the application Audit tab. In order to limit auditing to specific functions, set all properties of the /query/service record to NEVER and then enable audit for individual functions. If auditing for a function is enabled, all external calls to this function are audited. If audit of indirect calls is enabled, all calls originating from other data services are also audited.

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

Name	Enable Audit	Enable Audit of Indirect Calls
CUSTOMER	✓	☐
getADDRESS	☐	☐
getCREDIT_CARD	☐	☐
getCUSTOMER_ORDER	☐	☐

l. Click Apply. This enables auditing for the `CUSTOMER ()` function.

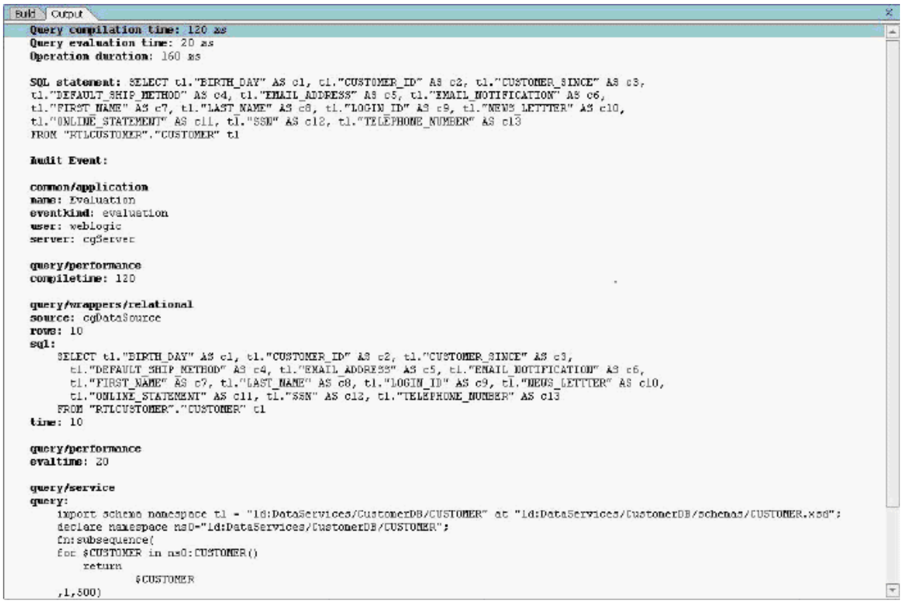
Notes:

- To enable auditing for any other function in this tutorial, repeat the steps h to l.
- Ensure that you keep auditing enabled in the ALDSP console throughout this tutorial. For details about auditing, refer to the Administration Guide.

6. In WebLogic Workshop → Test View, click Execute again.
7. Open the Output window (View → Windows → Output).
8. Confirm that the output is similar to that displayed in Figure 2-21.

Note: You can use the Output window to verify that each element in the data service is pulling data from the correct data source. In this example, the return results are pulled from the RTLCUSTOMER database, CUSTOMER table 1, and a specific column (c1, c2, c3, and so on) for each element.

Figure 2-21 Test Results Output



```
Query compilation time: 120 ms
Query evaluation time: 20 ms
Operation duration: 160 ms

SQL statement: 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

Audit Event:

common/application
name: Evaluation
eventkind: evaluation
user: weblogic
server: cgServer

query/performance
compiletime: 120

query/wrappers/relational
source: cgDataSource
rows: 10
sql:
    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
time: 10

query/performance
evaltime: 20

query/service
query:
    import schema namespace t1 = "ld:DataServices/CustomerDB/CUSTOMER" at "ld:DataServices/CustomerDB/schemas/CUSTOMER.xsd";
    declare namespace ns0="ld:DataServices/CustomerDB/CUSTOMER";
    fn:subsequence(
    for $CUSTOMER in ns0:CUSTOMER()
        return
                $CUSTOMER
    ,1,500)
```

Lesson Summary

In this lesson, you learned how to:

- Create a DSP application and project.
- Create project sub-folders to group data services.
- Import relational tables to create a simple physical data services.
- Build a project and review the build information.
- Examine a physical data service's shape/schema definition, data types, functions, and source code.
- Test a data service function.

Creating a Logical Data Service

As noted in Tutorial 2, there are two types of data services: physical and logical. Physical data services model a single physical data source residing in a relational database, Web service, flat file, XML file, or Java function.

To enable the integration of data from multiple sources through Data Services Platform (ALDSP), you define a logical data service. In this lesson you will create a logical data service that integrates data from the CUSTOMER data service.

Objectives

After completing this lesson, you will be able to:

- Create a simple logical data service, define its shape, and specify its query conditions
- Test the logical data service's read, write, and limit functions

Overview

A logical data service integrates data from two or more physical or logical data services. Its shape is defined by an XML type schema that classifies a data element as a particular form of information, according to its allowable contents and units of data. For example, an xs:string type can be a sequence of alphabetic, numeric, and/or special characters, while an xs:date type can only be numeric characters presented in a YYYY-MM-DD format.

The data service interface consists of public functions that enable client-based consuming applications to retrieve data from the modeled data source. A data service's functions can be of several types:

- One or more read functions, which typically return data in the form of the XML type.
- One or more navigate functions, which return data from related data services. Within a logical data service, you must define relationships through modeling. Although similar to relationships in the RDBMS context, a logical data service lets you establish relationships between data from any source. This gives you the ability to, for example, relate an ADDRESS relational table with a -STATE look-up Web service.
- One submit() function, which allow users to persist changes to the back-end storage

In addition to public functions, a data service can include private functions and side effect functions. Private functions are only used within the data service. They generally contain common processing logic that can be used by more than one data service function. Side effect functions can be invoked from the client side. For example, a side effect function can contain code to update a non-RDBMS data source, such as xml, flat files, and Web services, and clients can invoke this function to perform updates. (For more information, see the *Data Service Developer's Guide*.)

Every function within a logical data service also includes source-to-target mappings that define what results will be returned by that function. There are four types of mappings:

- A *simple mapping* means that you are mapping simple source node elements to simple elements in the Return type one at a time. You can create a simple mapping by dragging and dropping any element from the source node to its corresponding target element in the Return type. Optional Return type elements do not need to be mapped; otherwise elements in the Return type need to be mapped to run your query.
- An *induced mapping* means that a complex element is mapped to a complex element in the Return type. In this gesture, the top level complex element in the Return type is ignored (source node name need not match). The editor then automatically maps any child elements (complex or simple) that are an exact match for source node elements.
- An *overwrite mapping* replaces a Result type element and all its children (if any) with the source node elements. As an example of the general steps needed to create an overwrite mapping, you would press <Ctrl>, then drag and drop the source node's complex element onto the corresponding element in the Result type. The entire source node's complex element is brought to the Result type, where it completely replaces the target element with the source element.

- An *append mapping* adds a simple or complex element (and any children or attributes) as a child of the specified element in the Return type. To create an append mapping, select the source element, then press <Ctrl>+<Shift> while dragging and dropping the source node's element onto the element in the Return type that you want to be the parent of the new element(s).

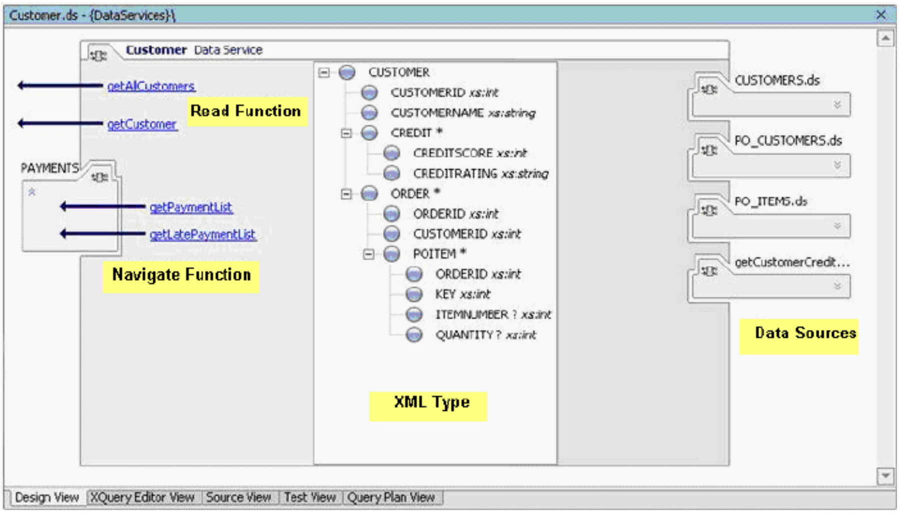
  Alternatively, if you simply want to add a child element to a Return type, you can drag a source element to a complex element in your Return type. The element will be added as a child of the complex element and mapped accordingly.

In addition to the mappings, each function can also include parameters and variations on the basic XQuery FLWOR (for-let-where-order by-return) statements that further define the data retrieval results.

When you click on the name of a data service in the Application pane (Figure 2-10), your data service will open in Design View (Figure 3-1). In the Customer data service, what you see in Design View is a logical data service that:

- Uses the `getAllCustomers()`, `getCustomer()`, `getPaymentList()`, and `getLatePaymentList()` functions to retrieve data.
- Uses the `customer.xsd` schema definition to define its XML type, and thus its Return type.
- Integrates data from the ApparelDB and CustomerDB physical data services, plus a CreditRating Web service.

Figure 3-1 Design View of a Logical Data Service



If you open XQuery Editor View for a particular function, you would see the function's source-to-target mappings.

If you open Source View, you would see each function's parameters and FLWOR statements.

3.1 Creating a Simple Logical Data Service

A logical data service integrates and transforms data from multiple physical and logical data services.

Objectives

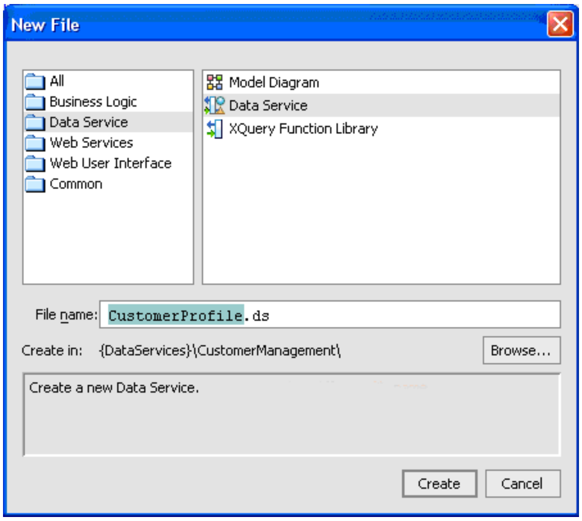
In this exercise, you will:

- Create a new folder for the logical data service.
- Create an empty data service that can be built into a logical data service.
- Import a pre-defined XML schema definition that you will associate as the logical data service's XML type.
- Define functions and their mappings, parameters, and FLWOR statements.

Instructions

1. Create a new folder within the DataServices project and name it CustomerManagement.
2. Create a new data service within the CustomerManagement folder by completing the following steps:
    a. Right-click the CustomerManagement folder.
    b. Choose New → Data Service. The New File dialog box opens.
    c. Confirm that Data Service → Data Service are selected.
    d. Enter CustomerProfile in the Name field.
    e. Click Create.

Figure 3-2 New Data Service



File name: CustomerProfile.ds

Create in: {DataServices}\CustomerManagement\

Create a new Data Service.

A new data service is generated, but without any associated data services or XML type.

3.2 Defining the Logical Data Service Shape

A data service transforms received data into the shape defined by its Return type. Pragmatically, the Return type is the "R" in a FLWOR (for-let-where-order by-return) query. A Return type, which describes the structure or shape of data returned by the data service's queries, serves two main purposes:

- Provides a superset of data elements that can be returned by an XQuery.
- Defines the unified structure, and order of the data returned by an XQuery.

The Return type is generated from the data service's XML type. An XML type classifies a data element as a particular form of information, according to its allowable contents and units of data. For example, an xs:string type can be a sequence of alphabetic, numeric, and/or special characters, while an xs:date type can only be numeric characters presented in a YYYY-MM-DD format.

Objectives

In this exercise, you will:

- Import a schema file, which you will associate with the data service's XML type.
- Review the results.

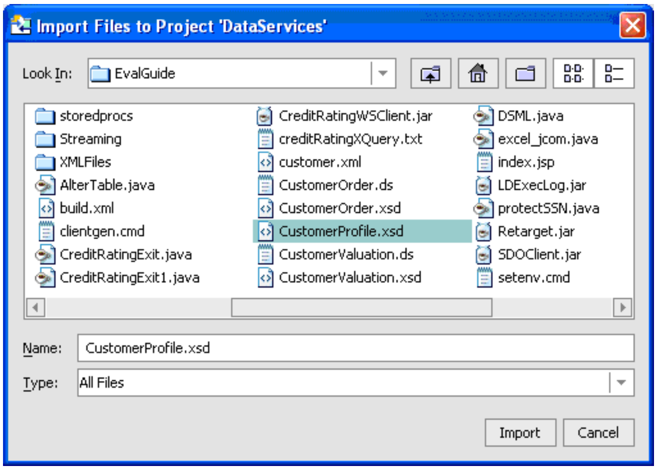
Instructions

Note: Although you can use ALDSP to graphically build a schema file, in this exercise you will import a pre-defined schema file to save time. For more information on using WebLogic Workshop to create the XML types, see the *Data Service Developer's Guide*.

1. Create a new folder in the CustomerManagement folder and name it schemas.
2. Import a schema file into the schema folder by completing the following steps:
    a. Right-click the schema folder, located in the CustomerManagement folder.
    b. Choose Import.
    c. Navigate to:

       `<beahome>\weblogic81\samples\liquiddata\EvalGuide`
    d. Select the `CustomerProfile.xsd` file.
    e. Click Import.

Figure 3-3 Import XML Schema Definition File



storedprocs
Streaming
XMLFiles
AlterTable.java
build.xml
clientgen.cmd
CreditRatingExit.java
CreditRatingExit1.java
CreditRatingWSClient.jar
creditRatingXQuery.txt
customer.xml
CustomerOrder.ds
CustomerOrder.xsd
CustomerProfile.xsd
CustomerValuation.ds
CustomerValuation.xsd
DSML.java
excel_jcom.java
index.jsp
LDExecLog.jar
protectSSN.java
Retarget.jar
SDOClient.jar
setenv.cmd

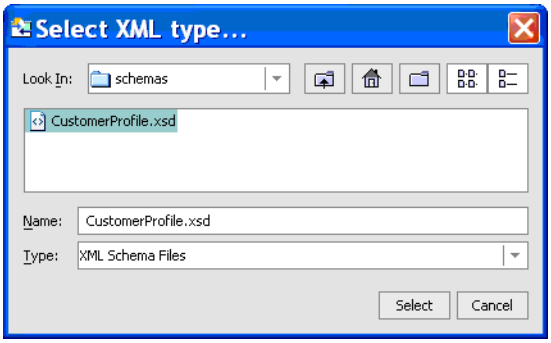
Name: CustomerProfile.xsd

Type: All Files

3. Right-click the CustomerProfile Data Service header on the Design View tab.
4. Choose Associate XML Type.
5. Select the `CustomerProfile.xsd` file, located in:

   `CustomerManagement\schemas`
6. Click Select.

Figure 3-4 Associating XML type with XSD



CustomerProfile.xsd

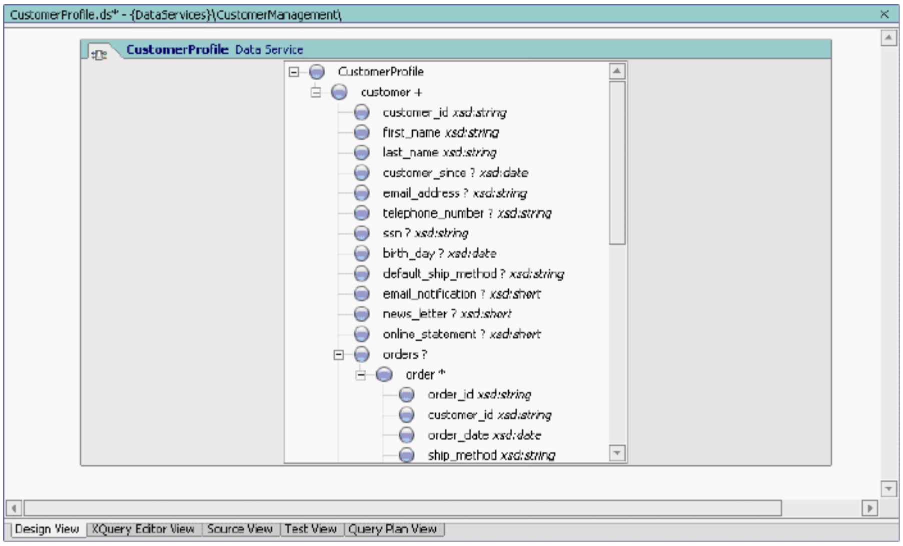
Name: CustomerProfile.xsd

Type: XML Schema Files

You should see that the CustomerProfile data service is now shaped by the `CustomerProfile.xsd` file.

You should also see that several of the elements are identified with a question (?) mark. This indicates that these elements are optional. Because the schema file identifies these elements as optional, ALDSP will not require the mapping of these elements to the Return type; however, if mapped to the Return type and there is no corresponding data in the underlying data source, then the result set will not include the empty elements.

Figure 3-5 Logical Data Service XML type


CustomerProfile.ds* - {DataServices}\CustomerManagement\
CustomerProfile Data Service
Design View
XQuery Editor View
Source View
Test View
Query Plan View

```
CustomerProfile
  customer +
    customer_id xsd:string
    first_name xsd:string
    last_name xsd:string
    customer_since ? xsd:date
    email_address ? xsd:string
    telephone_number ? xsd:string
    ssn ? xsd:string
    birth_day ? xsd:date
    default_ship_method ? xsd:string
    email_notification ? xsd:short
    news_letter ? xsd:short
    online_statement ? xsd:short
    orders ?
      order *
        order_id xsd:string
        customer_id xsd:string
        order_date xsd:date
        ship_method xsd:string
```

3.3 Adding a Function to a Logical Data Service

A data service consumer—a client application or another data service—uses the data service's function calls to retrieve information. A logical data service includes the same types of functions that are found in a physical data service:

- One or more read functions that form the data service's external interface, which is exposed to consuming applications requesting data. These read functions typically return data in the form of the data service's XML type.
- One or more navigate functions that return data from other data services. Within a logical data service, you must define relationships through modeling. Although similar to relationships in the RDBMS context, a logical data service lets you establish relationships between data from any source. This gives you the ability, for example, to relate an ADDRESS relational table with a STATE lookup Web service.

- One `submit()` function, which allows users to persist changes to the back-end storage.

Objectives

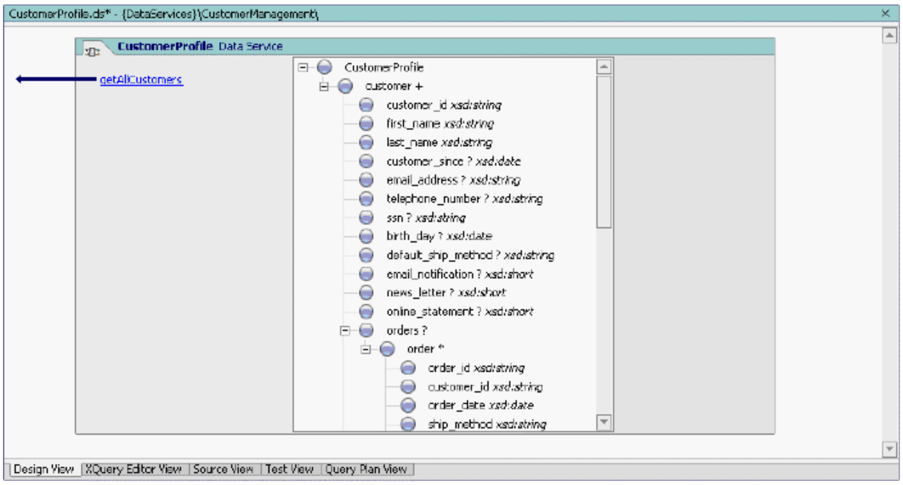
In this exercise, you will:

- Add a new read function, `getAllCustomers()`, to the logical data service.
- View the results in XQuery Editor View.

Instructions

1. Right-click the CustomerProfile Data Service header.
2. Choose Add Function. A new function displays in the left pane of the data service model.
3. Enter getAllCustomers as the function name.

Figure 3-6 Design View of New Function



3.4 Mapping Source and Target Elements

In the previous exercise, you associated a logical data service with an XML Schema Definition (`.xsd` file), which generated a Return type that includes all data elements defined within the schema.

However, there are no conditions associated with the Return type; conditions specify which source data will be returned.

You can define conditions by mapping source and target (Return) elements.

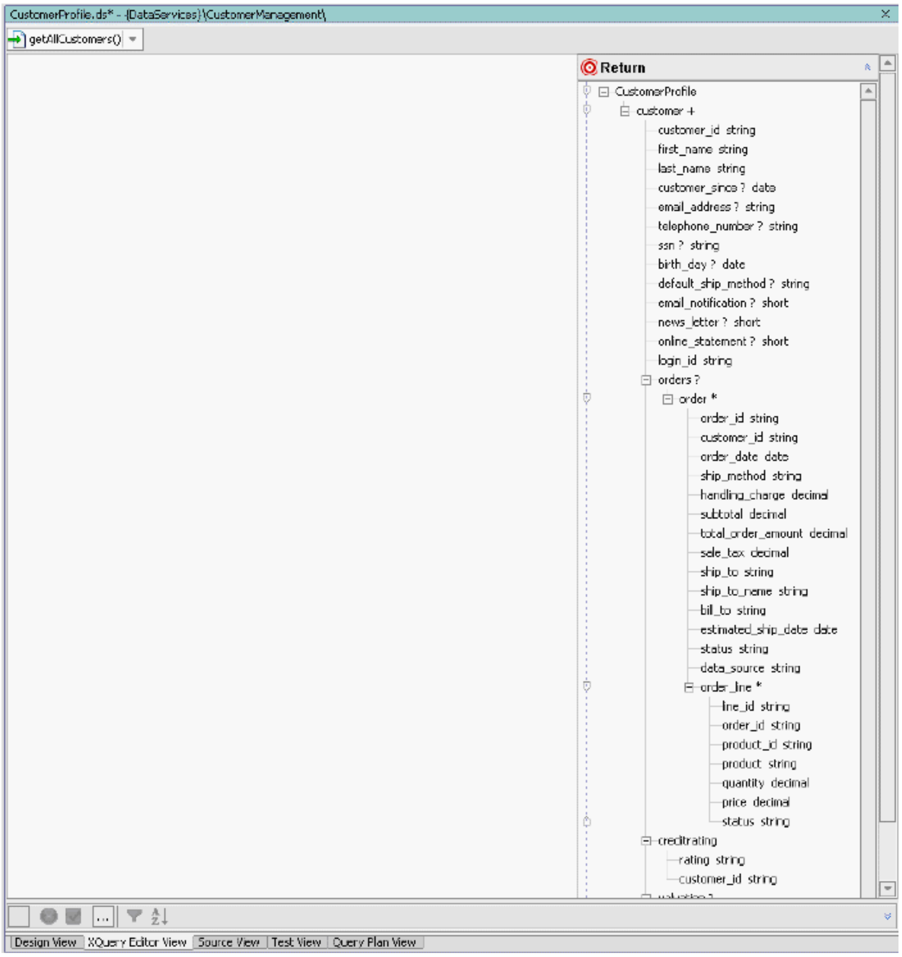
Objectives

- Add a physical data service function as a data source for the logical data service.
- Create a simple map between the source node and the Result type.

Instructions

1. Click the `getAllCustomers()` function to open XQuery Editor View. You should see a Return type populated with the CustomerProfile schema definition. The Return type determines what data can be made available to consuming applications, as well as the shape (string, data, integer, and so on) that the data will take. The Return type was automatically populated when you associated the logical data service with the `CustomerProfile.xsd`.

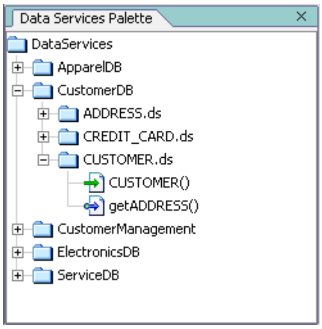
Figure 3-7 XQuery Editor View of Function Return Type



```
CustomerProfile
    customer +
        customer_id string
        first_name string
        last_name string
        customer_since ? date
        email_address ? string
        telephone_number ? string
        ssn ? string
        birth_day ? date
        default_ship_method ? string
        email_notification ? short
        news_letter ? short
        online_statement ? short
        login_id string
        orders ?
            order *
                order_id string
                customer_id string
                order_date date
                ship_method string
                handling_charge decimal
                subtotal decimal
                total_order_amount decimal
                sale_tax decimal
                ship_to string
                ship_to_name string
                bill_to string
                estimated_ship_date date
                status string
                data_source string
                order_line *
                    line_id string
                    order_id string
                    product_id string
                    product string
                    quantity decimal
                    price decimal
                    status string
        creditrating
            rating string
            customer_id string
```

2. In the Data Services Palette, expand `CustomerDB\CUSTOMER.ds`. If the Data Services Palette is not open, choose View → Windows → Data Services Palette.

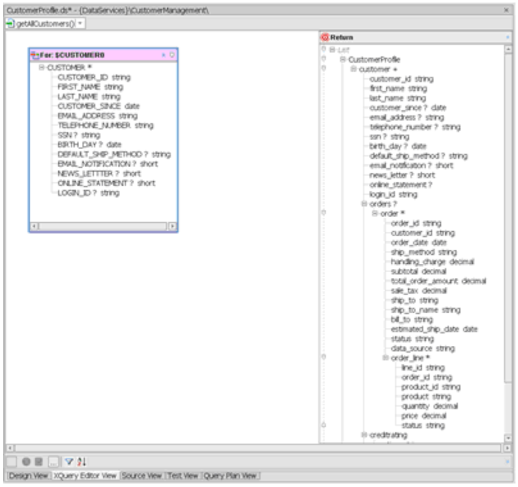
Figure 3-8 Data Services Palette



```
DataServices
  ApparelDB
  CustomerDB
    ADDRESS.ds
    CREDIT_CARD.ds
    CUSTOMER.ds
      CUSTOMER()
      getADDRESS()
  CustomerManagement
  ElectronicsDB
  ServiceDB
```

3. Drag and drop `CUSTOMER()` into XQuery Editor View. This method call represents a root or global element within the CUSTOMER physical data service (see 3.2 Defining the Logical Data Service Shape). A for node for that element is automatically generated and assigned a variable, such as For: $CUSTOMER. Within the XQuery Editor View, this for node is a graphical representation of a for clause, which is an integral part of an XQuery FLWOR expression (for-let-where-order by-return).

Figure 3-9 Source Node and Return Type



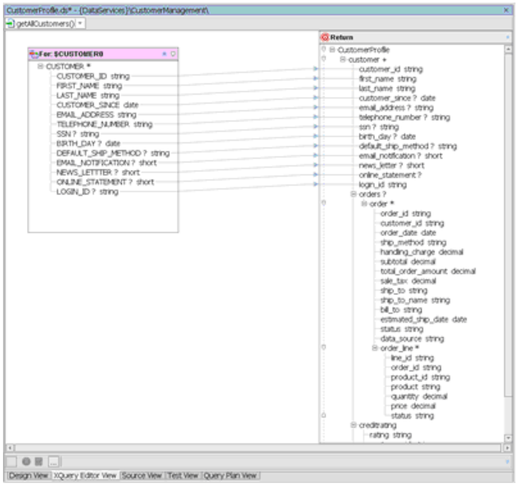
```
For: $CUSTOMER0
  CUSTOMER *
    CUSTOMER_ID  string
    FIRST_NAME  string
    LAST_NAME  string
    CUSTOMER_SINCE  date
    EMAIL_ADDRESS  string
    TELEPHONE_NUMBER  string
    SSN ?  string
    BIRTH_DAY ?  date
    DEFAULT_SHIP_METHOD ?  string
    EMAIL_NOTIFICATION ?  short
    NEWS_LETTTER ?  short
    ONLINE_STATEMENT ?  short
    LOGIN_ID ?  string
```

```
Return
  List
    CustomerProfile
      customer +
        customer_id  string
        first_name  string
        last_name  string
        customer_since ?  date
        email_address ?  string
        telephone_number ?  string
        ssn ?  string
        birth_day ?  date
        default_ship_method ?  string
        email_notification ?  short
        news_letter ?  short
        online_statement ?
        login_id  string
        orders ?
          order *
            order_id  string
            customer_id  string
            order_date  date
            ship_method  string
            handling_charge  decimal
            subtotal  decimal
            total_order_amount  decimal
            sale_tax  decimal
            ship_to  string
            ship_to_name  string
            bill_to  string
            estimated_ship_date  date
            status  string
            data_source  string
            order_line *
              line_id  string
              order_id  string
              product_id  string
              product  string
              quantity  decimal
              price  decimal
              status  string
        creditrating
```

4. Create a *simple* map by dragging and dropping individual elements from the $CUSTOMER source node onto the corresponding elements in the Return type. The logical data service CustomerProfile should now be similar to what is shown in Figure 3-10.

Note: There are alternatives to mapping elements instead of using the slow simple mapping technique. Faster mapping techniques are described in exercises that follow.

Figure 3-10 Simple Mapping Between Source Node and Return Type



3.5 Viewing XQuery Source Code

When you use XQuery Editor View to construct an XQuery, source code in XQuery syntax is automatically generated. You can view this generated source code in Source View and, if needed, modify the code. Any changes made in Source View will be reflected in XQuery Editor View.

Objectives

In this exercise, you will:

- View generated XQuery source code in Source View.
- Review the for and return clauses of the `getAllCustomers()` query function.

Instructions

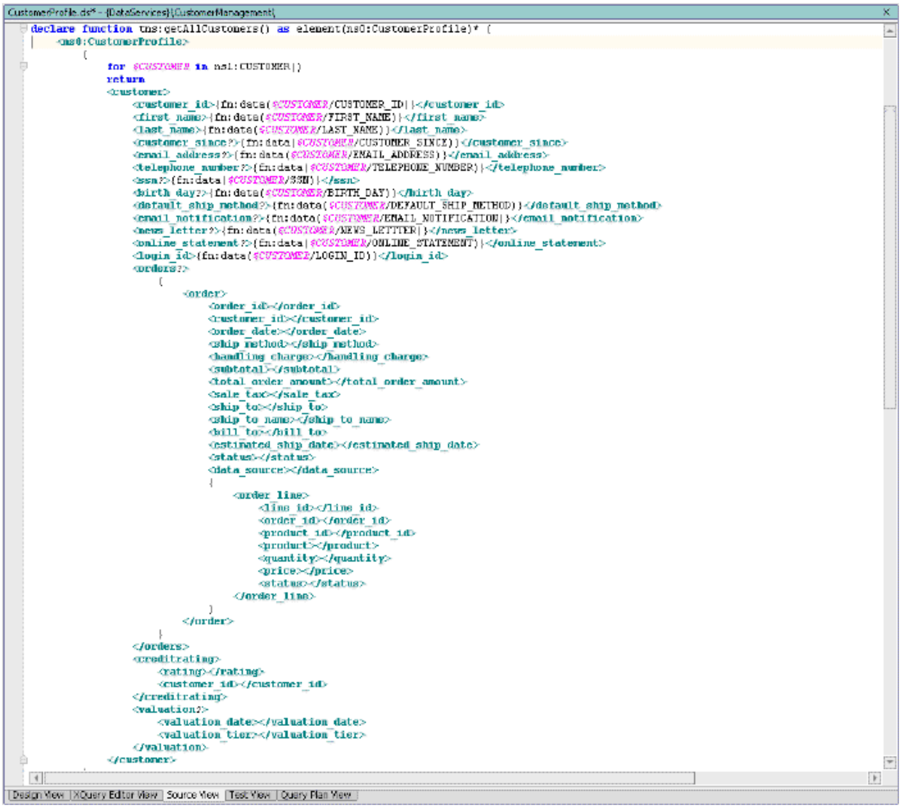
1. Select the Source View tab. A portion of the generated XQuery source code is displayed in Figure 3-11.
2. Notice the for clause, which references the `CUSTOMER()` function.

3. Notice the return clause, which reflects the simple mapping between the $CUSTOMER source node and the Return type. All optional elements are identified with a question mark in the field description, as shown (emphasis added):

```
<TelephoneNumber?> {fn:data(CUSTOMER/TELEPHONE_NUMBER)}</Telephone number
```

4. Also, notice that the <orders> elements are empty because order information has not yet been mapped to the Return type. This means that a consuming application, using this query, will only see customer information, not order information.

Figure 3-11 Source View of XQuery Code for CUSTOMER() Node



```
declare function tns:getAllCustomers() as element(ns0:CustomerProfile)* {
    <ns0:CustomerProfile>
        {
            for $CUSTOMER in ns1:CUSTOMER()
            return
            <customer>
                <customer_id>{fn:data($CUSTOMER/CUSTOMER_ID)}</customer_id>
                <first_name>{fn:data($CUSTOMER/FIRST_NAME)}</first_name>
                <last_name>{fn:data($CUSTOMER/LAST_NAME)}</last_name>
                <customer_since?>{fn:data($CUSTOMER/CUSTOMER_SINCE)}</customer_since>
                <email_address?>{fn:data($CUSTOMER/EMAIL_ADDRESS)}</email_address>
                <telephone_number?>{fn:data($CUSTOMER/TELEPHONE_NUMBER)}</telephone_number>
                <ssn?>{fn:data($CUSTOMER/SSN)}</ssn>
                <birth_day?>{fn:data($CUSTOMER/BIRTH_DAY)}</birth_day>
                <default_ship_method?>{fn:data($CUSTOMER/DEFAULT_SHIP_METHOD)}</default_ship_method>
                <email_notification?>{fn:data($CUSTOMER/EMAIL_NOTIFICATION)}</email_notification>
                <news_letter?>{fn:data($CUSTOMER/NEWS_LETTTER)}</news_letter>
                <online_statement?>{fn:data($CUSTOMER/ONLINE_STATEMENT)}</online_statement>
                <login_id>{fn:data($CUSTOMER/LOGIN_ID)}</login_id>
                <orders?>
                    {
                        <order>
                            <order_id></order_id>
                            <customer_id></customer_id>
                            <order_date></order_date>
                            <ship_method></ship_method>
                            <handling_charge></handling_charge>
                            <subtotal></subtotal>
                            <total_order_amount></total_order_amount>
                            <sale_tax></sale_tax>
                            <ship_to></ship_to>
                            <ship_to_name></ship_to_name>
                            <bill_to></bill_to>
                            <estimated_ship_date></estimated_ship_date>
                            <status></status>
                            <data_source></data_source>
                            {
                                <order_line>
                                    <line_id></line_id>
                                    <order_id></order_id>
                                    <product_id></product_id>
                                    <product></product>
                                    <quantity></quantity>
                                    <price></price>
                                    <status></status>
                                </order_line>
                            }
                        </order>
                    }
                </orders>
                <creditrating>
                    <rating></rating>
                    <customer_id></customer_id>
                </creditrating>
                <valuation?>
                    <valuation_date></valuation_date>
                    <valuation_tier></valuation_tier>
                </valuation>
            </customer>
```

3.6 Testing a Logical Data Service Function

You can use Test View to validate the functionality of a logical data service.

Objectives

In this exercise, you will:

- Build the DataServices project.
- Test the function's retrieve and limit result capabilities.

Instructions

1. Build the DataServices project by right-clicking the DataServices folder and choosing Build DataServices from the pop-up menu.
2. After the build completes successfully, select the Test View tab.
3. Select `getAllCustomers()` from the function drop-down list.

Test the ability to specify the number of tuples returned by completing the following steps:

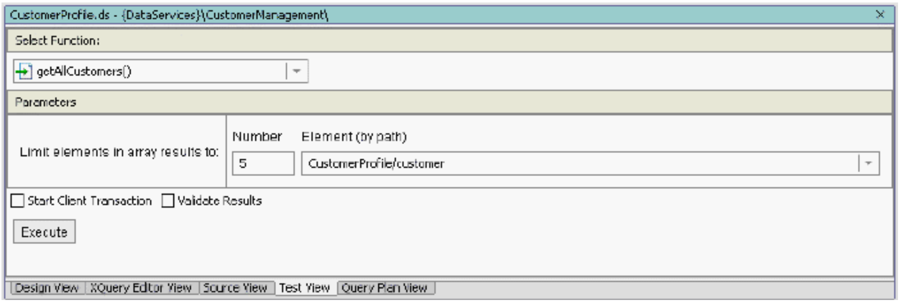
a. Uncheck the Validate Result option. This feature is not mandatory to complete this exercise.

b. Enter CustomerProfile/customer in the Parameter field (or select from the drop-down list). This parameter specifies the XPath expression for the element whose return results you want to limit to a set number of occurrences (such as customer).

c. Enter 5 in the Number field. This will limit the results to the first five customers retrieved.

d. Click Execute.

Figure 3-12 Test Truncate Capabilities


CustomerProfile.ds - {DataServices}\CustomerManagement\

Select Function: getAllCustomers()

Parameters

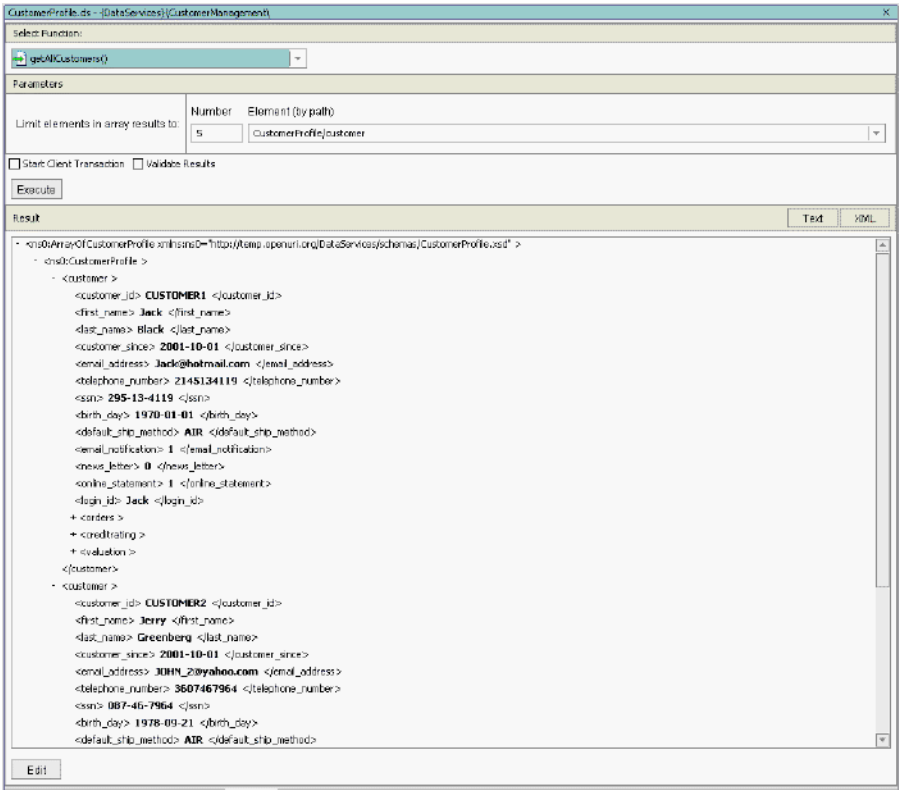
Limit elements in array results to: Number: 5 Element (by path): CustomerProfile/customer

☐ Start Client Transaction ☐ Validate Results

4. View the results, which appear in the Result pane.
5. Expand the top-level node. There should be only five Customer Profiles listed.

6. Expand the first <customer> node. You should see a Customer Profile for Jack Black, as displayed in Figure 3-13.

Figure 3-13 Customer Profile Test Results



Number: 5
Element (by path): CustomerProfile/customer

```
- <ns0:ArrayOfCustomerProfile xmlns:ns0="http://temp.openuri.org/DataServices/schemas/CustomerProfile.xsd" >
  - <ns0:CustomerProfile >
    - <customer >
        <customer_id> CUSTOMER1 </customer_id>
        <first_name> Jack </first_name>
        <last_name> Black </last_name>
        <customer_since> 2001-10-01 </customer_since>
        <email_address> Jack@hotmail.com </email_address>
        <telephone_number> 2145134119 </telephone_number>
        <ssn> 295-13-4119 </ssn>
        <birth_day> 1970-01-01 </birth_day>
        <default_ship_method> AIR </default_ship_method>
        <email_notification> 1 </email_notification>
        <news_letter> 0 </news_letter>
        <online_statement> 1 </online_statement>
        <login_id> Jack </login_id>
      + <orders >
      + <creditrating >
      + <valuation >
      </customer>
    - <customer >
        <customer_id> CUSTOMER2 </customer_id>
        <first_name> Jerry </first_name>
        <last_name> Greenberg </last_name>
        <customer_since> 2001-10-01 </customer_since>
        <email_address> JOHN_2@yahoo.com </email_address>
        <telephone_number> 3607467964 </telephone_number>
        <ssn> 087-46-7964 </ssn>
        <birth_day> 1978-09-21 </birth_day>
        <default_ship_method> AIR </default_ship_method>
```

Lesson Summary

In this lesson, you learned how to:

- Create a simple logical data service.
- Associate an XML schema definition with the data service.
- Create a simple function.
- Use XQuery editor view to map elements from the source node to the return type.

- Use Source View to examine an XQuery function’s source code.
- Use Test View to test a logical data service query capabilities, limit the number of data set results returned as part of the query, and test data service editing capabilities.

Integrating Data from Multiple Data Sources

The power of logical data services in Data Services Platform (ALDSP) is the ability to integrate and transform data from multiple physical and logical data services.

In the previous lesson, you created a simple logical data service that mapped to a single physical data service. In this lesson, you will further develop the logical data service to enable data retrieval from multiple data services.

Objectives

After completing this lesson, you will be able to:

- Use the Data Services Palette to add physical and logical data service functions to a logical data service, thereby accessing data from multiple sources.
- Join data services by connecting source elements, thereby integrating data from multiple sources.
- Use the Expression Builder to define a parameterized where clause.
- Create a complex overwrite mapping.
- Test parameterized data services to verify the return of integrated data results.

Overview

How is data integration different from process integration? Most applications involve a combination of informational interactions and transactional interactions. Examples of informational interaction include: get customer info, review order status, get customer profile, and get customer's case history. Examples of transactional interactions include: place order, update customer address, and create customer.

Informational interactions involve efficiently aggregating discrete pieces of data that are potentially resident in multiple data sources, and potentially in multiple data formats. Developers can end up spending inordinate amounts of time writing custom code to handle the various interface protocols and data formats, and integrate disparate data into manageable, business-relevant information. ALDSP simplifies this activity by providing a simple, declarative approach to aggregating data from heterogeneous data sources.

Transactional interactions involve taking a piece of data (say a purchase order) and orchestrating its propagation to the various underlying applications. This involves coordinating a business process through a formal or informal workflow, managing long-running processes, managing human interactions (such as a supervisor approval to an order), handling applications that have indeterminate response times (such as batch systems), maintaining transactional integrity across applications, etc.

Both data integration and process integration are essential elements when building applications that handle information from across multiple data sources. For functions of interest across data services, you can use function libraries. A function library (.xfl file) contains operations that return simple types (not the XML data type of a standard data service) that can be called from various data services. Read functions on a data service can be defined to return information in various ways. For example, the data service may define read functions for getting all customers, customers by region, or customers with a minimum order amount.

4.1 Joining Multiple Physical Data Services within a Logical Data Service

In the previous exercise, you mapped a single physical data service to the Return type. In this exercise, you will enable data retrieval from both the CUSTOMER and CUSTOMER_ORDER physical data services.

Objectives

In this exercise, you will:

- Create a second for node, by adding the `CUSTOMER_ORDER()` function to the XQuery Editor View.

- Create a simple map between the new for node and the Return type.
- Create an automatically-generated where clause, by joining the two for nodes.
- Review source code.
- Test the results (read and write capability)

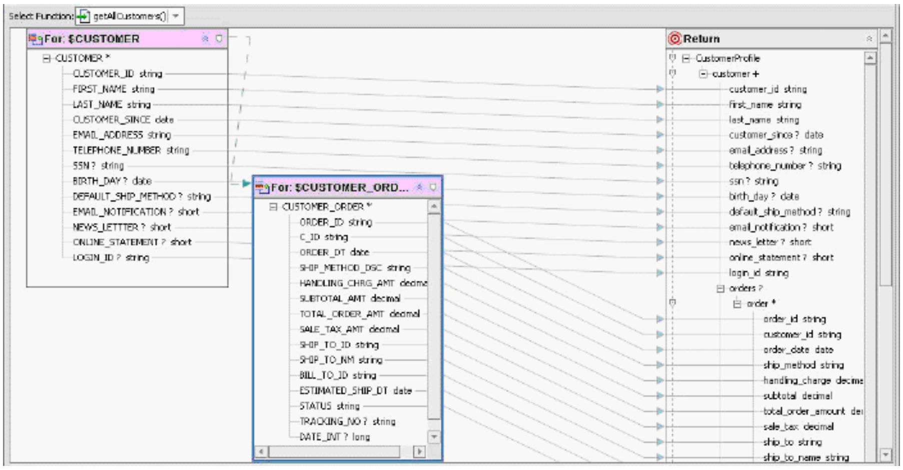
Instructions

1. Open CustomerProfile.ds in XQuery Editor View.
2. Select the `getAllCustomer()` function.
3. In the Data Services Palette, expand ApparelDB\CUSTOMER_ORDER data service.
4. Drag and drop the data service's `CUSTOMER_ORDER()` function into XQuery Editor View to create a second for node, For:$CUSTOMER_ORDER.
5. Create a simple map: Drag and drop the individual elements from the $CUSTOMER_ORDER source node onto their corresponding elements in the Return type.

Note: Do not map the TRACKING_NUMBER and DATE_INT elements.

6. Create a join: Drag and drop the CUSTOMER_ID element from the $CUSTOMER source node onto the C_ID element in the $CUSTOMER_ORDER source node. This action joins the two for nodes. By joining these two nodes, you automatically create a where clause within the FLWOR statement.

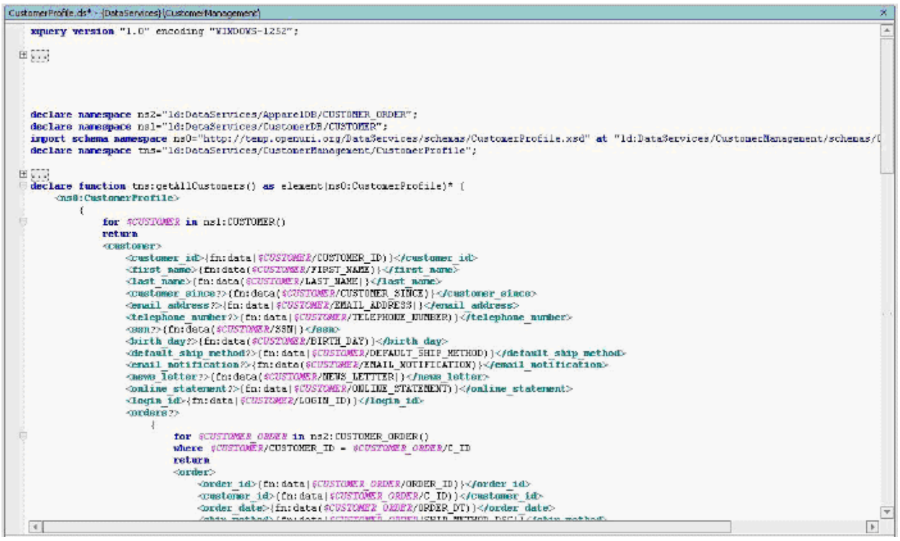
Figure 4-1 Joined Data Services



7. Select the Source View tab to view the XQuery code. You should see a where clause joining $CUSTOMER and $CUSTOMER_ORDER, using CUSTOMER_ID and C_ID as join elements. In Figure 4-2, the where clause is:

```
where $CUSTOMER/CUSTOMER_ID = $CUSTOMER_ORDER/C_ID
```

Figure 4-2 Source View of Joined Data Services



CustomerProfile.ds* - {DataServices}{CustomerManagement}

```
xquery version "1.0" encoding "WINDOWS-1252";

⊞ ...



declare namespace ns2="ld:DataServices/ApparelDB/CUSTOMER_ORDER";
declare namespace ns1="ld:DataServices/CustomerDB/CUSTOMER";
import schema namespace ns0="http://temp.openuri.org/DataServices/schemas/CustomerProfile.xsd" at "ld:DataServices/CustomerManagement/schemas/
declare namespace tns="ld:DataServices/CustomerManagement/CustomerProfile";

⊞ ...
declare function tns:getAllCustomers() as element(ns0:CustomerProfile)* {
    <ns0:CustomerProfile>
        {
            for $CUSTOMER in ns1:CUSTOMER()
            return
            <customer>
                <customer_id>{fn:data($CUSTOMER/CUSTOMER_ID)}</customer_id>
                <first_name>{fn:data($CUSTOMER/FIRST_NAME)}</first_name>
                <last_name>{fn:data($CUSTOMER/LAST_NAME)}</last_name>
                <customer_since?>{fn:data($CUSTOMER/CUSTOMER_SINCE)}</customer_since>
                <email_address?>{fn:data($CUSTOMER/EMAIL_ADDRESS)}</email_address>
                <telephone_number?>{fn:data($CUSTOMER/TELEPHONE_NUMBER)}</telephone_number>
                <ssn?>{fn:data($CUSTOMER/SSN)}</ssn>
                <birth_day?>{fn:data($CUSTOMER/BIRTH_DAY)}</birth_day>
                <default_ship_method?>{fn:data($CUSTOMER/DEFAULT_SHIP_METHOD)}</default_ship_method>
                <email_notification?>{fn:data($CUSTOMER/EMAIL_NOTIFICATION)}</email_notification>
                <news_letter?>{fn:data($CUSTOMER/NEWS_LETTTER)}</news_letter>
                <online_statement?>{fn:data($CUSTOMER/ONLINE_STATEMENT)}</online_statement>
                <login_id>{fn:data($CUSTOMER/LOGIN_ID)}</login_id>
                <orders?>
                    {
                        for $CUSTOMER_ORDER in ns2:CUSTOMER_ORDER()
                        where $CUSTOMER/CUSTOMER_ID = $CUSTOMER_ORDER/C_ID
                        return
                        <order>
                            <order_id>{fn:data($CUSTOMER_ORDER/ORDER_ID)}</order_id>
                            <customer_id>{fn:data($CUSTOMER_ORDER/C_ID)}</customer_id>
                            <order_date>{fn:data($CUSTOMER_ORDER/ORDER_DT)}</order_date>
```

8. Build the DataServices project. Right-click the DataServices project folder and choose Build DataServices.

9. After the build is successful, select the Test View tab in order to retrieve order information integrated with the customer information. You can do this by completing the following steps:

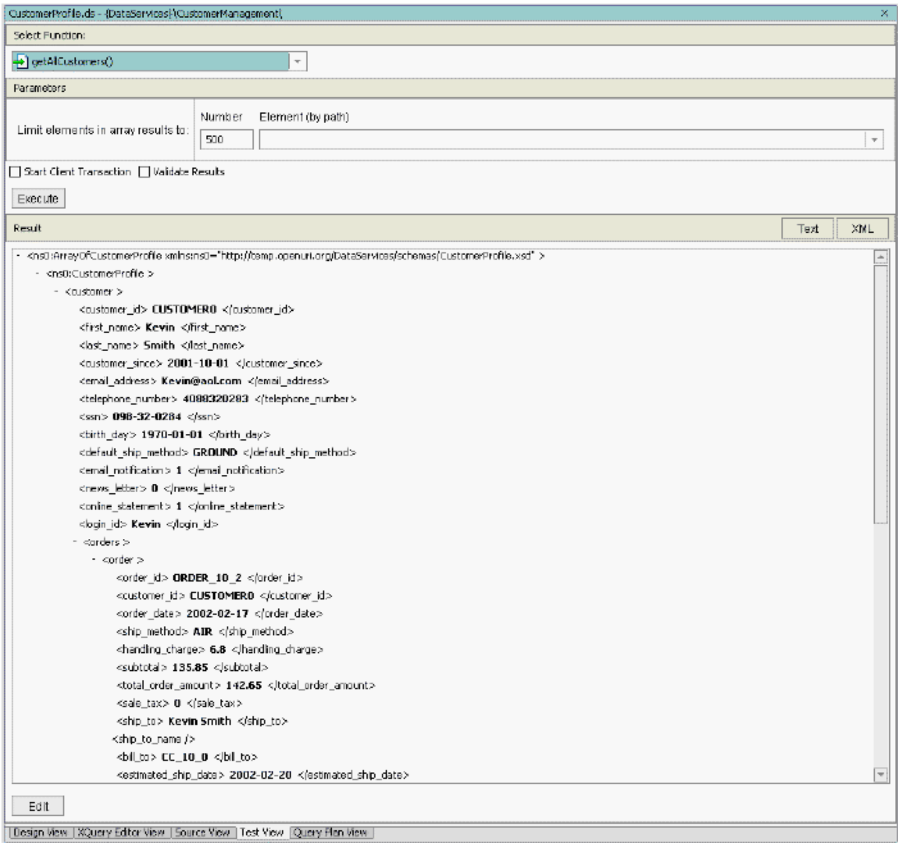
   a. Select `getAllCustomers()` from the function drop-down list.

   b. Click Execute. (You don't need any parameters, because you are not testing the limit returned tuples feature.)

   c. Expand the nodes. The results should include order information for each customer, as displayed in Figure 4-3.

Note: If the Validate Results option is selected, you will see a warning indicating that results do not conform to the associated XML type. The warning can be ignored.

Figure 4-3 Integrated Customer and Order Data Results



```
- <ns0:ArrayOfCustomerProfile xmlns:ns0="http://temp.openuri.org/DataServices/schemas/CustomerProfile.xsd" >
  - <ns0:CustomerProfile >
    - <customer >
        <customer_id> CUSTOMER0 </customer_id>
        <first_name> Kevin </first_name>
        <last_name> Smith </last_name>
        <customer_since> 2001-10-01 </customer_since>
        <email_address> Kevin@aol.com </email_address>
        <telephone_number> 4088320283 </telephone_number>
        <ssn> 098-32-0284 </ssn>
        <birth_day> 1970-01-01 </birth_day>
        <default_ship_method> GROUND </default_ship_method>
        <email_notification> 1 </email_notification>
        <news_letter> 0 </news_letter>
        <online_statement> 1 </online_statement>
        <login_id> Kevin </login_id>
      - <orders >
        - <order >
            <order_id> ORDER_10_2 </order_id>
            <customer_id> CUSTOMER0 </customer_id>
            <order_date> 2002-02-17 </order_date>
            <ship_method> AIR </ship_method>
            <handling_charge> 6.8 </handling_charge>
            <subtotal> 135.85 </subtotal>
            <total_order_amount> 142.65 </total_order_amount>
            <sale_tax> 0 </sale_tax>
            <ship_to> Kevin Smith </ship_to>
           <ship_to_name />
            <bill_to> CC_10_0 </bill_to>
            <estimated_ship_date> 2002-02-20 </estimated_ship_date>
```

4.2 Defining a Where Clause to Join Multiple Physical Data Services

In the previous exercise, you joined the CUSTOMER and CUSTOMER_ORDER data services, thereby automatically generating a where clause. In this exercise, you will manually define the where clause that joins multiple data services.

Objectives

In this exercise, you will:

- Add a third for node, by adding the `CUSTOMER_ORDER_LINE_ITEM()` function.
- Define a where clause, using the Expression Editor.
- View the results in Design View and Source View.
- Test the results.

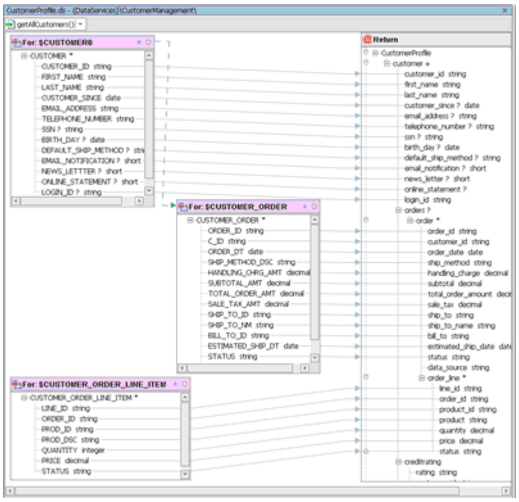
Instructions

1. Switch to XQuery Editor View for the `getAllCustomers()` function.
2. In the Data Services Palette, expand ApparelDB\CUSTOMER_ORDER_LINE_ITEM data services.
3. Drag and drop the `CUSTOMER_ORDER_LINE_ITEM()` function from the Data Service palette into the data service's XQuery Editor View. This creates a third for node:

`For: $CUSTOMER_ORDER_LINE_ITEM.`

4. Create simple mappings by dragging and dropping the individual elements from the $CUSTOMER_ORDER_LINE_ITEM source node onto the corresponding elements in the Return type.

Figure 4-4 Three Data Service Functions Mapped to the Return Type



CustomerProfile.ds - {DataServices}\CustomerManagement\
getAllCustomers()
For: $CUSTOMERS
For: $CUSTOMER_ORDER
For: $CUSTOMER_ORDER_LINE_ITEM
Return

5. Define a where clause for CUSTOMER_ORDER and CUSTOMER_ORDER_LINE_ITEM, by completing the following steps:

a. Select the node header (For: $CUSTOMER_ORDER_LINE_ITEM) to activate the expression editor for that node. (Note: Do not select the CUSTOMER_ORDER_LINE_ITEM* element.)

b. Click the Where clause icon.

c. Put your cursor into the where expression line editor.

d. Click the ORDER_ID element in the $CUSTOMER_ORDER_LINE_ITEM source node. You should see the following in the WHERE field (the variable name may be different, in your case):

```
$CUSTOMER_ORDER_LINE_ITEM/ORDER_ID
```

e. Select eq: Compare Single Values from the operator list ("…" icon). Since the Where clause is incomplete, the text will go red. The Where field now appears as:

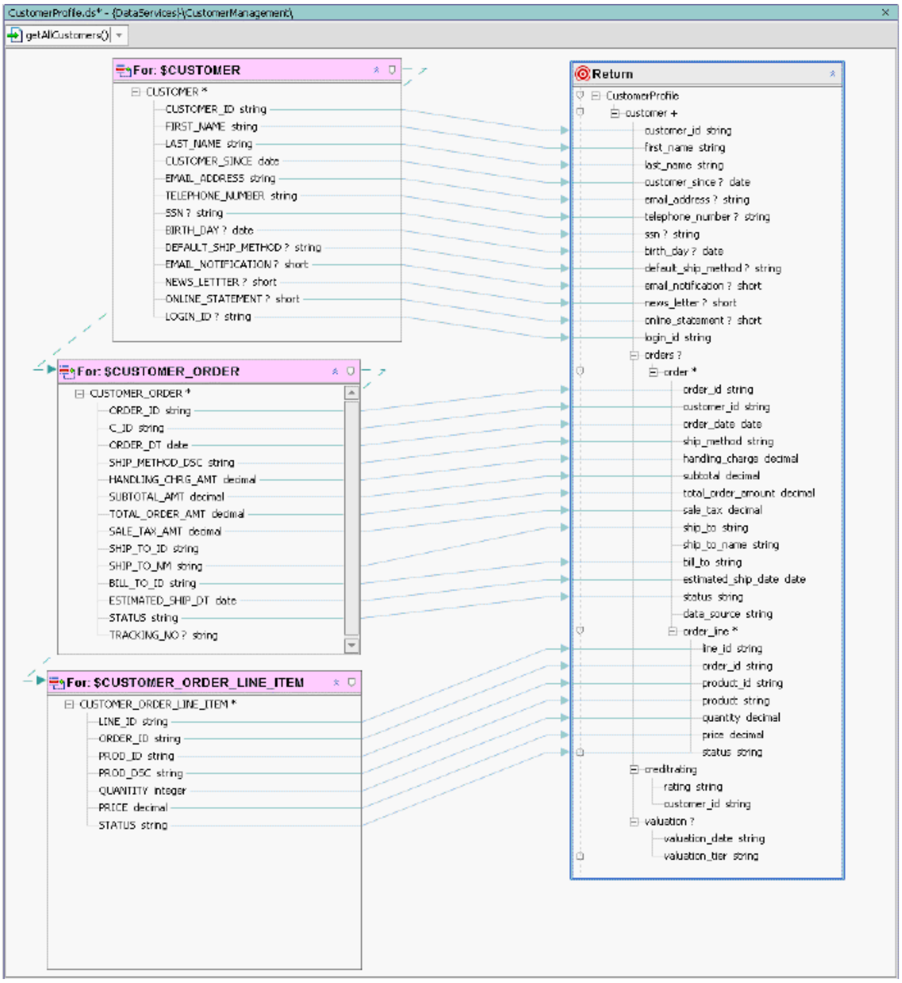
```
$CUSTOMER_ORDER_LINE_ITEM/ORDER_ID eq
```

f. Click the ORDER_ID element in the CUSTOMER_ORDER source node. The Where clause becomes valid and you should see the following in the where field (the variable name may be different, in your case):

```
$CUSTOMER_ORDER_LINE_ITEM/ORDER_ID eq $CUSTOMER_ORDER/ORDER_ID
```

g. Click the Accept box (green checkmark icon) to add the parameterized WHERE clause to the `getAllCustomers()` function.

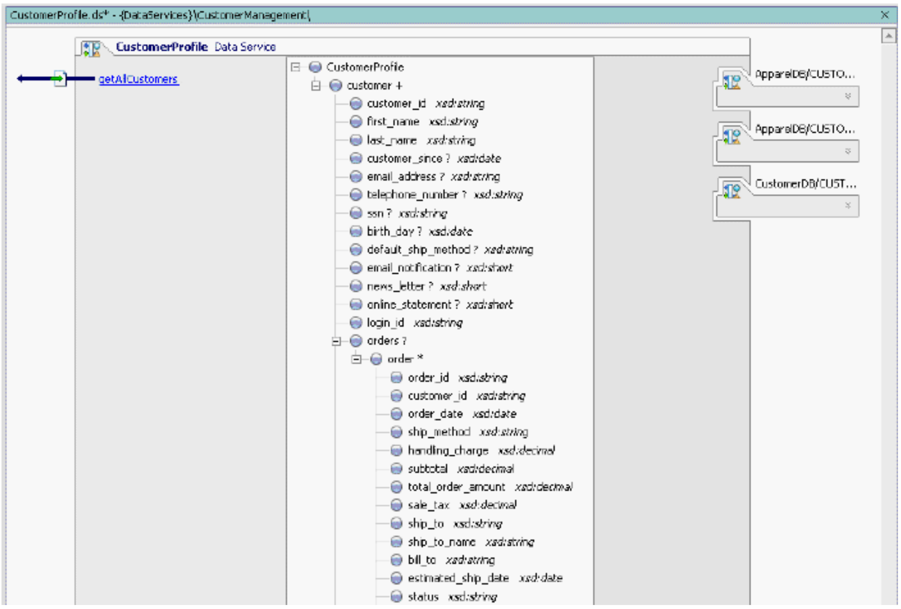
Figure 4-5 Where Clause Joining Two Data Services



6. Verify the joins you created and view the results by completing the following steps:

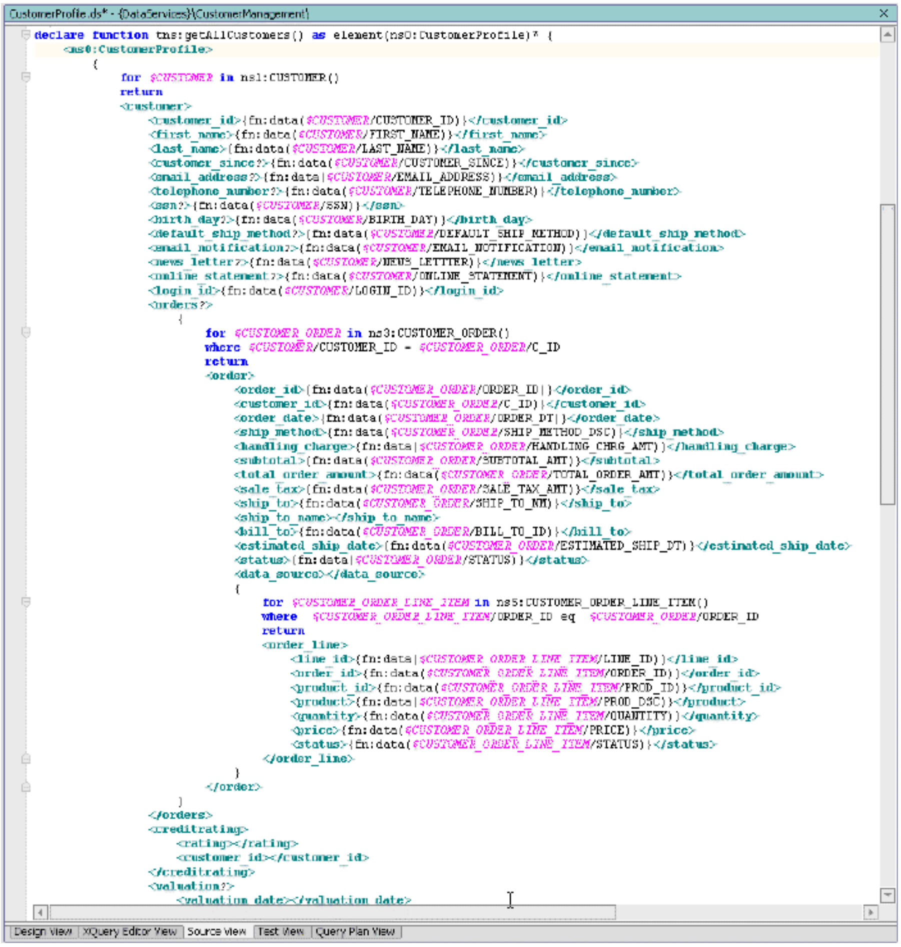
   a. Open CustomerProfile.ds in Design View. The physical data services associated with the three functions that you dropped into XQuery Editor View as for nodes are displayed in the right pane as data sources for the logical data service.

Figure 4-6 Design View of Integrated and Parameterized Data Service



b. Open `CustomerProfile.ds` in Source View. The XQuery code for the logical data service is displayed.

Figure 4-7 Source Code for Data Integrated with WHERE Clauses and Parameters



```
declare function tns:getAllCustomers() as element(ns0:CustomerProfile)* {
    <ns0:CustomerProfile>
        {
            for $CUSTOMER in ns1:CUSTOMER()
            return
            <customer>
                <customer_id>{fn:data($CUSTOMER/CUSTOMER_ID)}</customer_id>
                <first_name>{fn:data($CUSTOMER/FIRST_NAME)}</first_name>
                <last_name>{fn:data($CUSTOMER/LAST_NAME)}</last_name>
                <customer_since?>{fn:data($CUSTOMER/CUSTOMER_SINCE)}</customer_since>
                <email_address?>{fn:data($CUSTOMER/EMAIL_ADDRESS)}</email_address>
                <telephone_number?>{fn:data($CUSTOMER/TELEPHONE_NUMBER)}</telephone_number>
                <ssn?>{fn:data($CUSTOMER/SSN)}</ssn>
                <birth_day?>{fn:data($CUSTOMER/BIRTH_DAY)}</birth_day>
                <default_ship_method?>{fn:data($CUSTOMER/DEFAULT_SHIP_METHOD)}</default_ship_method>
                <email_notification?>{fn:data($CUSTOMER/EMAIL_NOTIFICATION)}</email_notification>
                <news_letter?>{fn:data($CUSTOMER/NEWS_LETTTER)}</news_letter>
                <online_statement?>{fn:data($CUSTOMER/ONLINE_STATEMENT)}</online_statement>
                <login_id>{fn:data($CUSTOMER/LOGIN_ID)}</login_id>
                <orders?>
                    {
                        for $CUSTOMER_ORDER in ns3:CUSTOMER_ORDER()
                        where $CUSTOMER/CUSTOMER_ID = $CUSTOMER_ORDER/C_ID
                        return
                        <order>
                            <order_id>{fn:data($CUSTOMER_ORDER/ORDER_ID)}</order_id>
                            <customer_id>{fn:data($CUSTOMER_ORDER/C_ID)}</customer_id>
                            <order_date>{fn:data($CUSTOMER_ORDER/ORDER_DT)}</order_date>
                            <ship_method>{fn:data($CUSTOMER_ORDER/SHIP_METHOD_DSC)}</ship_method>
                            <handling_charge>{fn:data($CUSTOMER_ORDER/HANDLING_CHRG_AMT)}</handling_charge>
                            <subtotal>{fn:data($CUSTOMER_ORDER/SUBTOTAL_AMT)}</subtotal>
                            <total_order_amount>{fn:data($CUSTOMER_ORDER/TOTAL_ORDER_AMT)}</total_order_amount>
                            <sale_tax>{fn:data($CUSTOMER_ORDER/SALE_TAX_AMT)}</sale_tax>
                            <ship_to>{fn:data($CUSTOMER_ORDER/SHIP_TO_NM)}</ship_to>
                            <ship_to_name></ship_to_name>
                            <bill_to>{fn:data($CUSTOMER_ORDER/BILL_TO_ID)}</bill_to>
                            <estimated_ship_date>{fn:data($CUSTOMER_ORDER/ESTIMATED_SHIP_DT)}</estimated_ship_date>
                            <status>{fn:data($CUSTOMER_ORDER/STATUS)}</status>
                            <data_source></data_source>
                            {
                                for $CUSTOMER_ORDER_LINE_ITEM in ns5:CUSTOMER_ORDER_LINE_ITEM()
                                where  $CUSTOMER_ORDER_LINE_ITEM/ORDER_ID eq  $CUSTOMER_ORDER/ORDER_ID
                                return
                                <order_line>
                                    <line_id>{fn:data($CUSTOMER_ORDER_LINE_ITEM/LINE_ID)}</line_id>
                                    <order_id>{fn:data($CUSTOMER_ORDER_LINE_ITEM/ORDER_ID)}</order_id>
                                    <product_id>{fn:data($CUSTOMER_ORDER_LINE_ITEM/PROD_ID)}</product_id>
                                    <product>{fn:data($CUSTOMER_ORDER_LINE_ITEM/PROD_DSC)}</product>
                                    <quantity>{fn:data($CUSTOMER_ORDER_LINE_ITEM/QUANTITY)}</quantity>
                                    <price>{fn:data($CUSTOMER_ORDER_LINE_ITEM/PRICE)}</price>
                                    <status>{fn:data($CUSTOMER_ORDER_LINE_ITEM/STATUS)}</status>
                                </order_line>
                            }
                        </order>
                    }
                </orders>
                <creditrating>
                    <rating></rating>
                    <customer_id></customer_id>
                </creditrating>
                <valuation?>
                    <valuation_date></valuation_date>
```

7. Test the results, by completing the following steps:

    a. Build the DataServices project.

    b. Open `CustomerProfile.ds` in Test View.

    c. Select `getAllCustomers()` from the function drop-down list.

    d. Set the element (by path) option to CustomerProfile/customer.

e. Click Execute. (You do not need any parameters.)

f. Expand the nodes and confirm that you can retrieve order line information integrated with order information, similar to that displayed in Figure 4-8. (You can use customer_id = CUSTOMER3 to verify this information).

g. Click Edit.

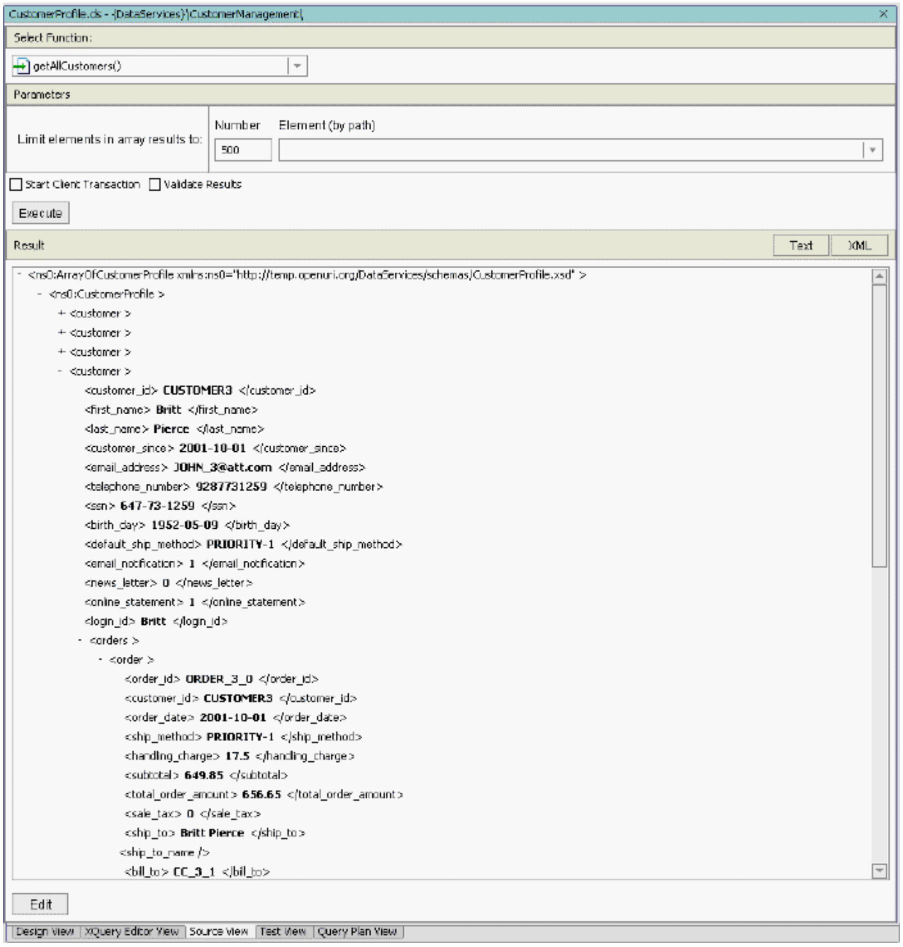
h. Navigate to the Orders node for CUSTOMER3 and update handling_charge information for ORDER_3_0 by double clicking the element content (the 6.8 value).

i. Change to any value other than the current value.

j. Confirm your new value by pressing Submit button.

k. Verify that the update was done successfully by re-executing `getAllCustomers()` function and navigating to order information for CUSTOMER3.

Figure 4-8 Order Line Data Integrated Withing Order Information



```
- <ns0:ArrayOfCustomerProfile xmlns:ns0="http://temp.openuri.org/DataServices/schemas/CustomerProfile.xsd" >
  - <ns0:CustomerProfile >
    + <customer >
    + <customer >
    + <customer >
    - <customer >
        <customer_id> CUSTOMER3 </customer_id>
        <first_name> Britt </first_name>
        <last_name> Pierce </last_name>
        <customer_since> 2001-10-01 </customer_since>
        <email_address> JOHN_3@att.com </email_address>
        <telephone_number> 9287731259 </telephone_number>
        <ssn> 647-73-1259 </ssn>
        <birth_day> 1952-05-09 </birth_day>
        <default_ship_method> PRIORITY-1 </default_ship_method>
        <email_notification> 1 </email_notification>
        <news_letter> 0 </news_letter>
        <online_statement> 1 </online_statement>
        <login_id> Britt </login_id>
      - <orders >
        - <order >
            <order_id> ORDER_3_0 </order_id>
            <customer_id> CUSTOMER3 </customer_id>
            <order_date> 2001-10-01 </order_date>
            <ship_method> PRIORITY-1 </ship_method>
            <handling_charge> 17.5 </handling_charge>
            <subtotal> 649.85 </subtotal>
            <total_order_amount> 656.65 </total_order_amount>
            <sale_tax> 0 </sale_tax>
            <ship_to> Britt Pierce </ship_to>
           <ship_to_name />
            <bill_to> CC_3_1 </bill_to>
```

4.3 Creating a Parameterized Function

Adding a parameter to a function ensures that the consuming application can access specific user-defined data, such as an individual customer's profile information.

Objectives

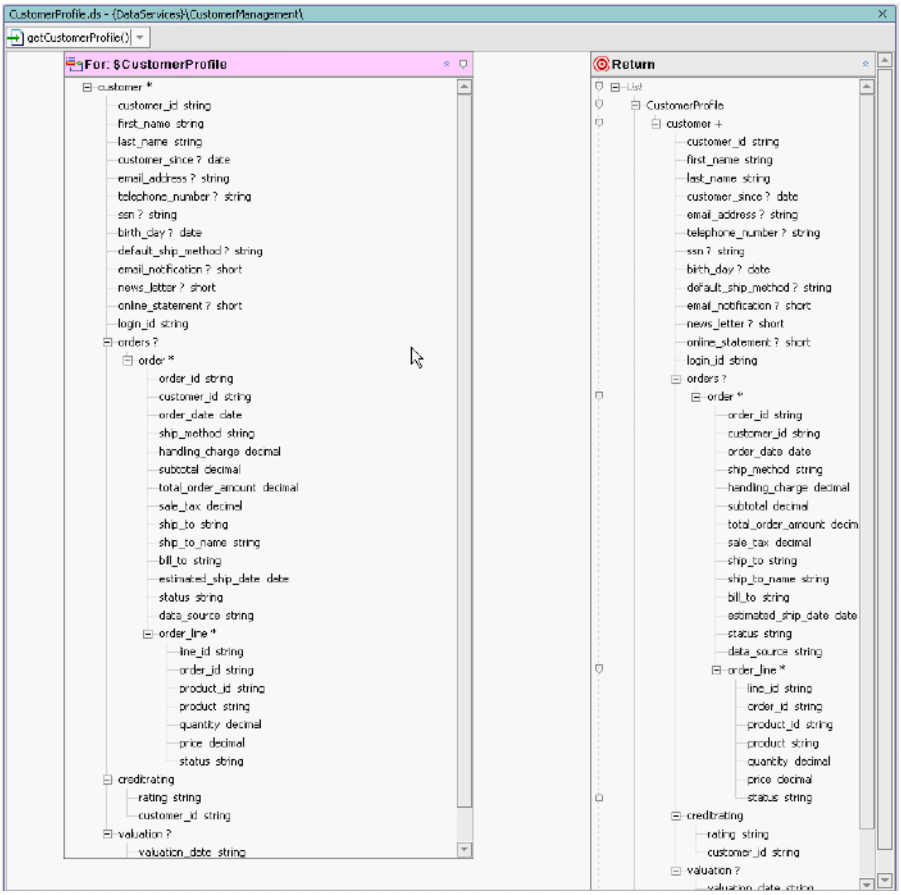
In this exercise, you will:

- Add a new function, `getCustomerProfile()`.
- Add a for node based on the `getAllCustomers()` function.
- Set the context for nested elements within the logical data service.

Instructions

1. In Design View, create a new function for the CustomerProfile data service, and name it `getCustomerProfile()`.
2. Click `getCustomerProfile()` to open XQuery Editor View for that function.
3. In the Data Services Palette, expand CustomerManagement\CustomerProfile data service.
4. Drag and drop `getAllCustomers()` into the XQuery Editor View. You should see a new for node. For: $CustomerProfile, with its shape defined by the CustomerProfile logical data service's `getAllCustomers()` function.

Figure 4-9 Complex Element Node



Note: In a previous exercise, you defined `getAllCustomers()` to include a complex, nested customer element associated with the customer_id element of the $CUSTOMER_ORDER_LINE_ITEM source. You must set the context of the $CustomerProfile source node to point to the customer element because customer_id uses a string parameter for filtering.

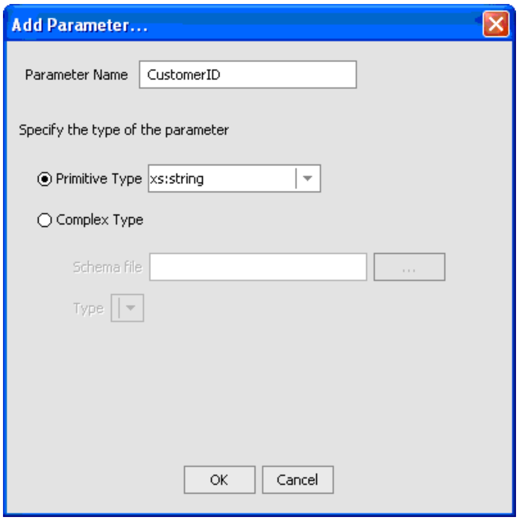
5. Create a parameter by completing the following steps:
   a. Right-click an empty space in XQuery Editor View.
   b. Select Add Parameter.

c. Enter CustomerID in the Parameter Name field.

d. Select xs:string from the Primitive Type drop-down list.

e. Click OK.

Figure 4-10 Add Parameter



Note: You may need to move the $CustomerProfile node to make the parameter node visible.

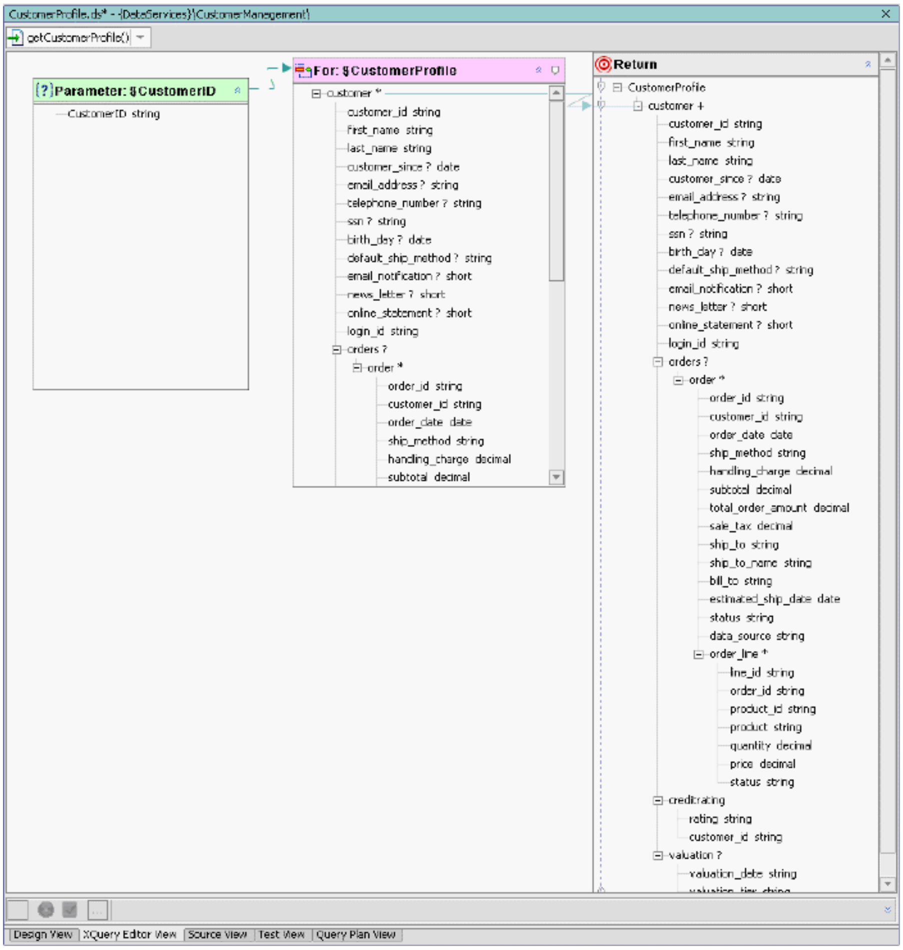
6. Create a complex, overwrite mapping, by completing the following steps:

   a. Press Ctrl.

   b. Drag and drop the $CustomerProfile customer* element onto the customer+ element in the Return type.

7. Create a join: Drag and drop the parameter's string element onto the customer_id element of the $CustomerProfile source node. This joins the string parameter to the $CustomerProfile source node and creates a function that will return data based on the user-specified parameter. (You will see this in action in the next exercise.)

Figure 4-11 Data Source Node and Parameter Joined

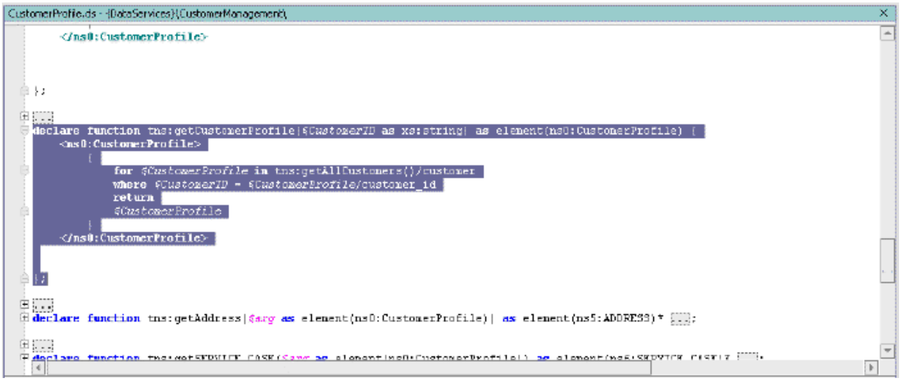


8. Select the Source View tab and confirm that the XQuery code for the `getCustomerProfile()` function is as follows:

```
declare function tns:getCustomerProfile($CustomerID as xs:string) as
element(ns0:CustomerProfile)* {

    <ns0:CustomerProfile>

        {
            for $CustomerProfile in tns:getAllCustomers()/customer
            where $CustomerID = $CustomerProfile/customer_id
            return
            $CustomerProfile
        }
    </ns0:CustomerProfile>
```

9. Remove the asterisk * from the return type element(ns0:CustomerProfile)*, because this function, as currently written, will return all customer profiles. The exercise calls for returning a single customer profile. Thus your source should be similar to that displayed in Figure 4-12.

Figure 4-12 Source Code for a Parameterized and Complex Overwrite Mapped Function



CustomerProfile.ds - {DataServices}\CustomerManagement\

```
    </ns0:CustomerProfile>


};

...
declare function tns:getCustomerProfile($CustomerID as xs:string) as element(ns0:CustomerProfile) {
    <ns0:CustomerProfile>
        {
            for $CustomerProfile in tns:getAllCustomers()/customer
            where $CustomerID = $CustomerProfile/customer_id
            return
            $CustomerProfile
        }
    </ns0:CustomerProfile>

};

...
declare function tns:getAddress($arg as element(ns0:CustomerProfile)) as element(ns5:ADDRESS)* ...;

...
declare function tns:getSERVICE_CASE($arg as element(ns0:CustomerProfile)) as element(ns6:SERVICE_CASE)* ...;
```

10. Test the function, by completing the following steps:

    a. Build your project.

    b. Open `CustomerProfile.ds` in Test View.

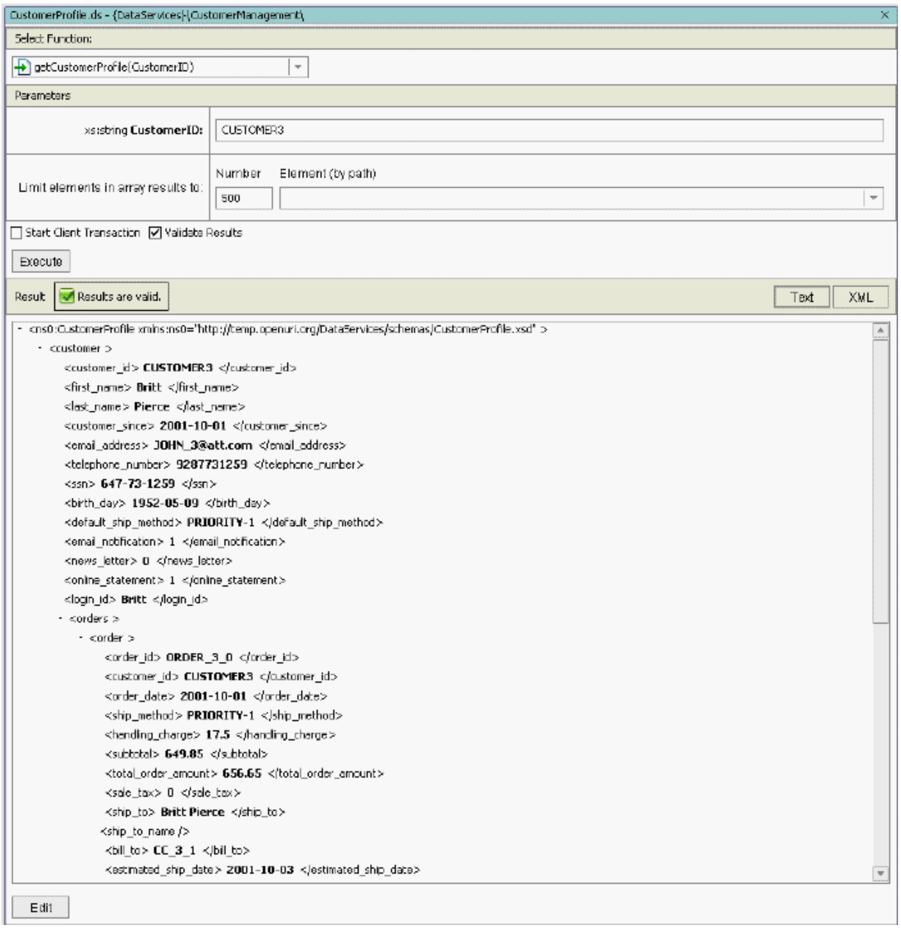
    c. Select `getCustomerProfile(CustomerID)` from the function drop-down list.

    d. Enter CUSTOMER3 in the xs:string CustomerID Parameter field. (Note: The parameter is case-sensitive.)

    e. Press Execute.

    f. Confirm that you retrieved the requested information — customer, orders, and order line items for Britt Pierce.

Figure 4-13 Integrated Data Results



CustomerProfile.ds - {DataServices}-\CustomerManagement\
Select Function:
getCustomerProfile(CustomerID)
Parameters
Execute
Text
XML
Edit

xs:string **CustomerID:** CUSTOMER3

Limit elements in array results to: Number 500 Element (by path)

Start Client Transaction [✓ Validate Results]

Result: Results are valid.

```
- <ns0:CustomerProfile xmlns:ns0="http://temp.openuri.org/DataServices/schemas/CustomerProfile.xsd" >
  - <customer >
      <customer_id> CUSTOMER3 </customer_id>
      <first_name> Britt </first_name>
      <last_name> Pierce </last_name>
      <customer_since> 2001-10-01 </customer_since>
      <email_address> JOHN_3@att.com </email_address>
      <telephone_number> 9287731259 </telephone_number>
      <ssn> 647-73-1259 </ssn>
      <birth_day> 1952-05-09 </birth_day>
      <default_ship_method> PRIORITY-1 </default_ship_method>
      <email_notification> 1 </email_notification>
      <news_letter> 0 </news_letter>
      <online_statement> 1 </online_statement>
      <login_id> Britt </login_id>
    - <orders >
      - <order >
          <order_id> ORDER_3_0 </order_id>
          <customer_id> CUSTOMER3 </customer_id>
          <order_date> 2001-10-01 </order_date>
          <ship_method> PRIORITY-1 </ship_method>
          <handling_charge> 17.5 </handling_charge>
          <subtotal> 649.85 </subtotal>
          <total_order_amount> 656.65 </total_order_amount>
          <sale_tax> 0 </sale_tax>
          <ship_to> Britt Pierce </ship_to>
         <ship_to_name />
          <bill_to> CC_3_1 </bill_to>
          <estimated_ship_date> 2001-10-03 </estimated_ship_date>
```

Lesson Summary

In this lesson, you learned how to:

- Use the Data Services Palette to add physical and logical data service functions to a logical data service, thereby accessing data from multiple sources.
- Join data services by connecting source elements, thereby integrating data from multiple sources.

- Use the Expression Builder to define a parameterized where clause.
- Set the context for nested elements in the source node.
- Create a complex overwrite mapping.
- Test parameterized data service function to verify the return of integrated data results.

Modeling Data Services

Any data service — physical or logical — can be placed in a model diagram. Model diagrams show:

- The basic structure of data returned by each data service within the model.
- Any functions associated with that data service.
- Any relationships between data services.

The main purpose of the diagram is to help you envision meaningful subsets of the model, but it can also be used to define new artifacts or edit existing artifacts.

Objectives

After completing this lesson, you will be able to:

- Create model diagrams and add data source nodes to the diagram.
- Confirm relationships inferred during the Import Source Metadata process.
- Define new relationships between data services and modify relationship properties.

Overview

Model diagrams show how various data services are related. Models can represent physical data services, logical data services, or a combination.

Each physical model entity represents a single data source. In the case of relational sources, you can automatically generate physical models that are representative of data sources. After being generated, physical data services can be integrated with other physical or logical sources in the same or new models. Physical model types use a key icon to identify primary keys.

Logical data model entities, which are discussed in detail in the Data Service Developer's Guide, represent composite views of physical and/or logical models.

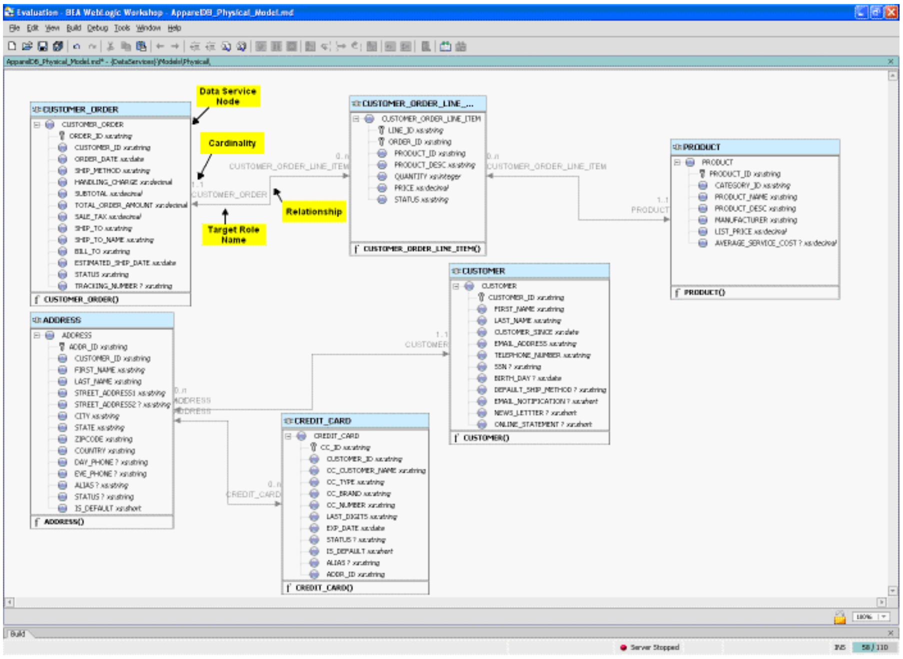
Within the model diagram, data services appear as boxes. Relationships are represented by annotated lines between two data services. Each side of the relationship line represents the role played by the nearest data service. The annotations for each relationship include the following:

- **Target Role Name.** By default, the target role name reflects the name of its adjacent data service. You can modify the target role name to better express the relationship, which is particularly useful when there are multiple relationships between two data services.
- **Cardinality.** A relationship can be zero-to-one (0:1 or 1:0), one-to-one (1:1), one-to-many (1:*n*) or many-to-many (n:n). For example, a customer can have multiple orders, therefore, the relationship should be 1:n (customer:orders).
- **Directionality.** A relationship can be either unidirectional or bidirectional. If unidirectional, data service a can navigate to data service b but b does not navigate to a. If bidirectional, data service a can navigate to b and b can navigate to a.

A data service's navigation functions determine the relationship's cardinality and directionality. Arrowheads indicate possible navigation paths.

ALDSP model diagrams are very flexible; they can be based on existing data services (and corresponding underlying data sources), planned data services, or a combination. Using models you can easily manage multiple data services as well as identify needs for new data services. You can also create and modify data service types directly in the modeler and inspect data services.

Figure 5-1 Model Diagram for Physical Data Services



5.1 Creating a Basic Model Diagram for Physical Data Services

Modeling data services begins by adding individual data services to a diagram.

Objectives

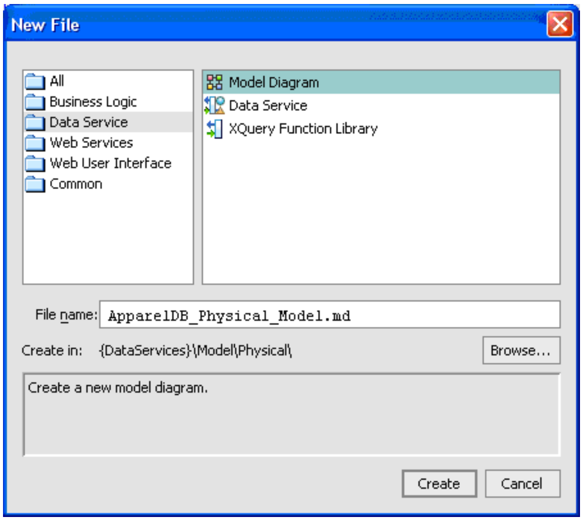
In this exercise, you will:

- Create a diagram that you will use to model relationships between physical data services.
- Add the ApparelDB and CustomerDB physical data services to the model diagram.
- Confirm relationships "captured" during the Import Source Metadata process.

Instructions

1. Create a new folder in the DataServices project and name it Models.
2. Create a new folder in the Models folder and name it Physical.
3. Create a blank model diagram, by completing the following steps:
    a. Right-click the Physical folder.
    b. Choose New → Model Diagram.
    c. Select Data Service → Model Diagram as shown in Figure 5-2.

Figure 5-2 Create Model Diagram



File name: ApparelDB_Physical_Model.md

Create in: {DataServices}\Model\Physical\

Create a new model diagram.

    d. Enter ApparelDB_Physical_Model in the File name field.
    e. Click Create. A blank workspace opens. You can use that workspace to construct your model diagram.

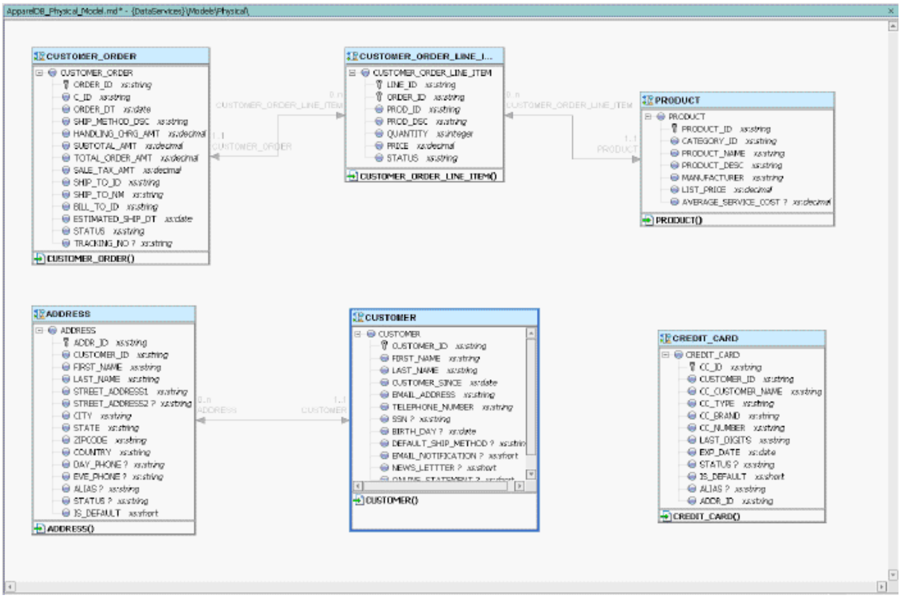
4. Add the ApparelDB and CustomerDB physical data services to the model by dragging and dropping the following data service files from the Application pane into the model:

Data Service File	Location

CUSTOMER_ORDER.ds	DataServices\ApparelDB
CUSTOMER_ORDER-LINE_ITEM.ds	DataServices\ApparelDB
PRODUCT.ds	DataServices\ApparelDB
ADDRESS.ds	DataServices\CustomerDB
CREDIT_CARD.ds	DataServices\CustomerDB
CUSTOMER.ds	DataServices\CustomerDB

Notice that relationships between some data services already exist. These relationships were automatically generated during the Import Source Metadata process, and are based on the foreign key relationships defined in the underlying database.

Figure 5-3 Physcial Data Services Model Diagram



5.2 Modeling Relationships Between Physical Data Sources

The next step in data service modeling is to define additional relationships, beyond any relationship that was automatically generated during the import source metadata process.

A relationship is a logical connection between two data services, such as the CUSTOMER and CUSTOMER_ORDER data services. A relationship exists when one data service retrieves data from another, by invoking one or more of the other data service's functions.

- A data service's navigation functions determine the relationship's cardinality and directionality. Arrowheads indicate possible navigation paths. Directionality can be either one directional or bidirectional.

Objectives

In this exercise, you will:

- Define a relationship between the CUSTOMER and CUSTOMER_ORDER nodes, thereby creating a navigational function between the two nodes.
- Modify the relationship properties to enable a "1:0 or many" relationship.

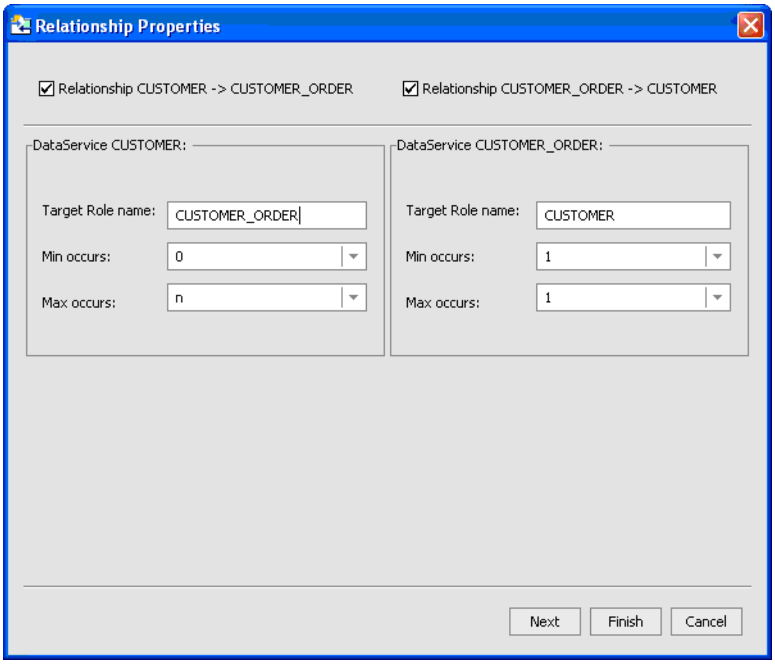
Instructions

1. Drag and drop the top-level CUSTOMER element onto the top-level CUSTOMER_ORDER element. The Relationship Properties dialog box opens.
2. In the Relationship Properties dialog box, modify the cardinality properties of the CUSTOMER and CUSTOMER_ORDER data services, by completing the following steps for the CUSTOMER node:
   a. Select 0 from the Min occurs drop-down list.
   b. Select n from the Max occurs drop-down list.

   The relationship cardinality is now "1:0 or many" between the CUSTOMER and CUSTOMER_ORDER data services. In other words, one customer can have none, one, or any number of orders.
3. Click Finish.

Note: In subsequent lessons, you will use additional features of the Relationship Properties dialog box to customize relationship properties.

Figure 5-4 Relationship Properties -- Cardinality



[✓] Relationship CUSTOMER -> CUSTOMER_ORDER
[✓] Relationship CUSTOMER_ORDER -> CUSTOMER

DataService CUSTOMER:

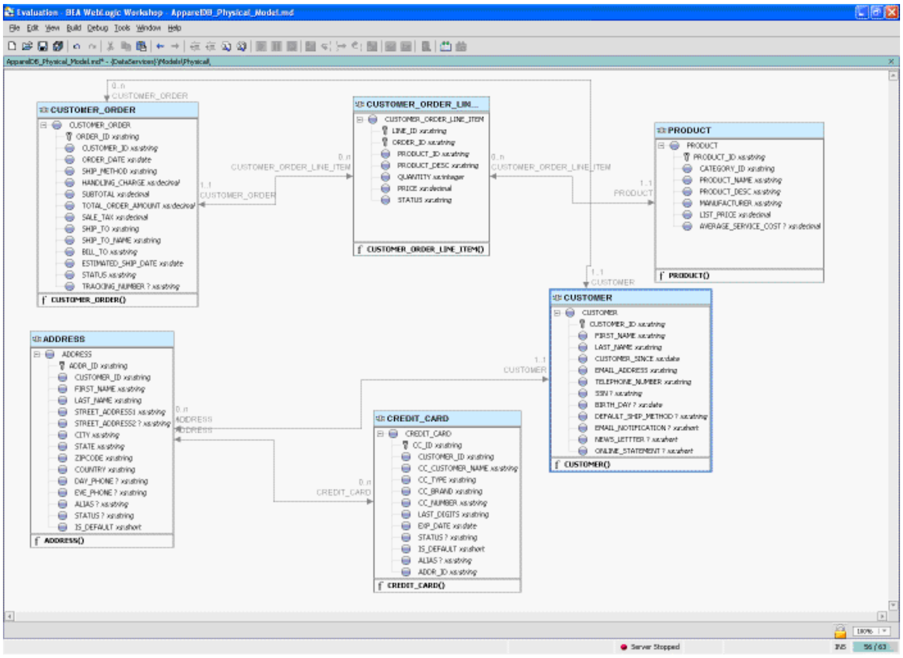
Target Role name: CUSTOMER_ORDER
Min occurs: 0
Max occurs: n

DataService CUSTOMER_ORDER:

Target Role name: CUSTOMER
Min occurs: 1
Max occurs: 1

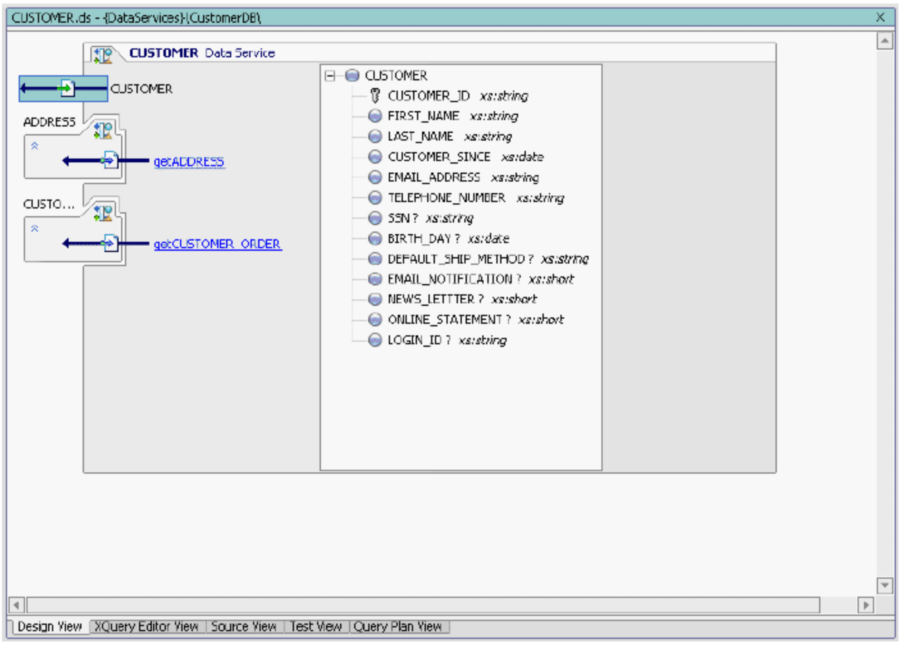
Note: It may take a few seconds to generate the relationship line.

Figure 5-5 New Relationship Between Customer and Customer_Order Data Services Defined



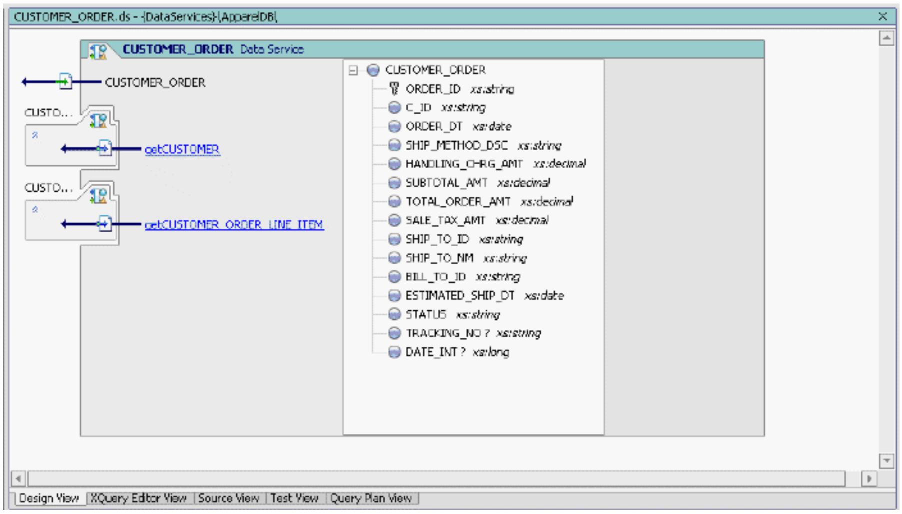
4. Save all your files using the File → Save All command.
5. Open `CUSTOMER.ds` in Design View. The file is located in the DataServices\CustomerDB folder.
6. Confirm that the CUSTOMER data service includes a new relationship with the CUSTOMER_ORDER data service, using the `getCustomer_Order()` function.

Figure 5-6 CUSTOMER Data Service Showing Added Relationship Function



7. Open `CUSTOMER_ORDER.ds` in Design View. The file is located in DataServices\ApparelDB.
8. Confirm that the CUSTOMER_ORDER data service includes a new relationship with the CUSTOMER data service, using the `getCustomer()` function.

Figure 5-7 CUSTOMER_ORDER Data Service Showing Added Relationship Function



```
CUSTOMER_ORDER
  ORDER_ID  xs:string
  C_ID  xs:string
  ORDER_DT  xs:date
  SHIP_METHOD_DSC  xs:string
  HANDLING_CHRG_AMT  xs:decimal
  SUBTOTAL_AMT  xs:decimal
  TOTAL_ORDER_AMT  xs:decimal
  SALE_TAX_AMT  xs:decimal
  SHIP_TO_ID  xs:string
  SHIP_TO_NM  xs:string
  BILL_TO_ID  xs:string
  ESTIMATED_SHIP_DT  xs:date
  STATUS  xs:string
  TRACKING_NO ?  xs:string
  DATE_INT ?  xs:long
```

9. (*Optional*) Create a relationship between CUSTOMER and CREDIT_CARD data services.

10. (*Optional*) Close all open files.

Lesson Summary

In this lesson, you learned how to:

- Create model diagrams and add data source nodes to the diagram.
- Confirm relationships inferred during the Import Source Metadata process.
- Define relationships between data services.

Accessing Data Services

One of the data sources available with the samples installed with ALDSP is a Web service that provides customer credit rating information. In this lesson, you will generate a physical data service that can be integrated into the CustomerProfile logical data service.

The process for creating a data service based on a Web service is similar to importing relational database source metadata. The difference is that ALDSP uses the WSDL (Web services description language) metadata to introspect the Web service's operation and generate the data service.

Objectives

After completing this lesson, you will be able to:

- Import a WSDL.
- Use the WSDL to generate a data service.
- Test the Web service by passing a SOAP request body as a query parameter.
- Use a logical data service to invoke the Web service and retrieve data.

Overview

A Web service is a self-contained, platform-independent unit of business logic that is accessible to other systems on a network. The network can be a corporate intranet or the Internet. Other systems can call the Web services' functions to request data or perform an operation.

Web services are increasingly important resources for global business information. Web services can facilitate application-to-application communication and are a useful way to provide data, like stock quotes and weather reports, to an array of consumers over a corporate intranet or the Internet. But they take on additional new power in the enterprise, where they offer a flexible solution for integrating distributed systems, whether legacy systems or new technology.

WSDLs are generally publicly accessible and provide enough detail so that potential clients can figure out how to operate the service solely from reading the WSDL file. If a Web service translates English sentences into French, the WSDL file will explain how the English sentences should be sent to the Web service, and how the French translation will be returned to the requesting client.

6.1 Importing a Web Service Project into the Application

When you want to use an external Web service from within WebLogic Workshop, you should first obtain that service's WSDL file. In this exercise, you will use the WSDL for a Web service project that was created in WebLogic Workshop.

Objectives

In this exercise, you will:

- Import the CreditRatingWS Web service into your sample application. This Web service provides `getCreditRating()` and `setCreditRating()` functions for retrieving and updating a customer's credit rating.
- Run the Web service to test whether you can retrieve credit rating information.

Instructions

1. Import a Web service into the ALDSP-enabled application, by completing the following steps:
   a. Choose File → Import Project. The Import Project - New Project dialog box opens.
   b. Select Web Service Project.

Caution: Make sure that you select a project of type Web service. If you select another project type, then the CreditRatingWS application may not work correctly.

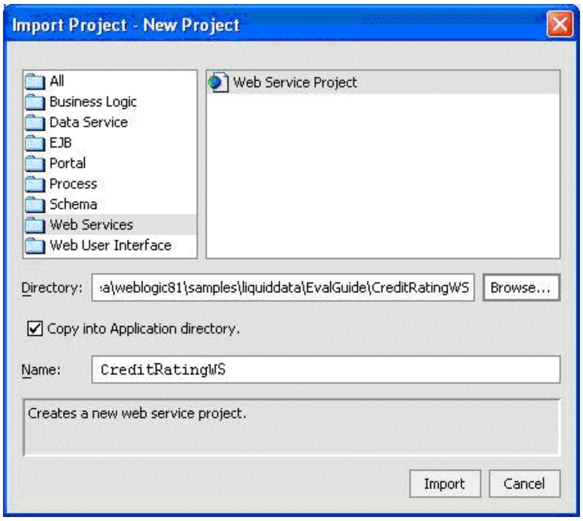
   c. In the directory field, click Browse.
   d. Navigate to <beahome>\weblogic81\samples\liquiddata\EvalGuide

e. Select CreditRatingWS and click Open.

f. Make sure that the Copy into Application directory checkbox is selected.

g. Click Import and then click Yes when the confirmation message to update your project appears.

Figure 6-1 Import Web Services Project



Directory: :a\weblogic81\samples\liquiddata\EvalGuide\CreditRatingWS

☑ Copy into Application directory.

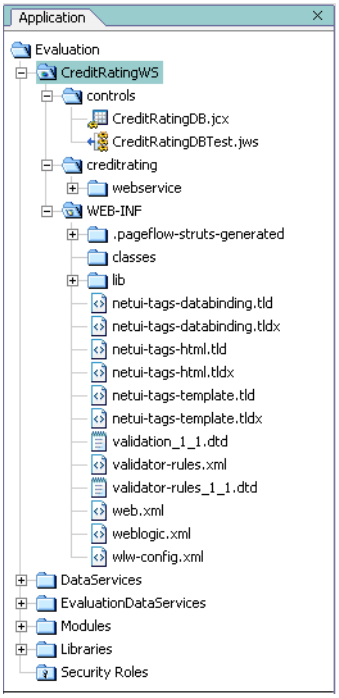
Name: CreditRatingWS

Creates a new web service project.

2. In the Application pane, verify that the following items were imported:

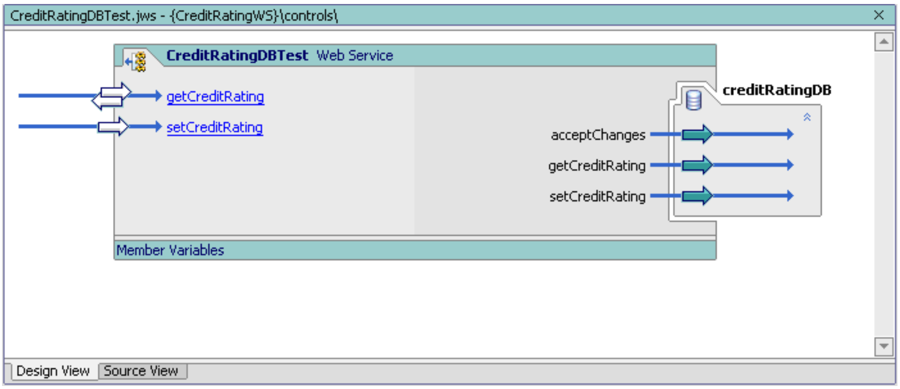
- A CreditRatingWS project folder containing:
  - A controls folder, within which are the `CreditRatingDB.jcx` control and `CreditratingDBTest.jws` Web service.
  - A credit rating folder, within which is the Web service folder that contains the `CreditRating.java` file.
  - A WEB-INF folder.

Figure 6-2 Web Service Project



3. Open `CreditRatingDBTest.jws` in Design View. This file is located in CreditRatingWS\controls. The Web service diagram should be as displayed in Figure 6-3.

Figure 6-3 Design View of Credit Rating Web Service



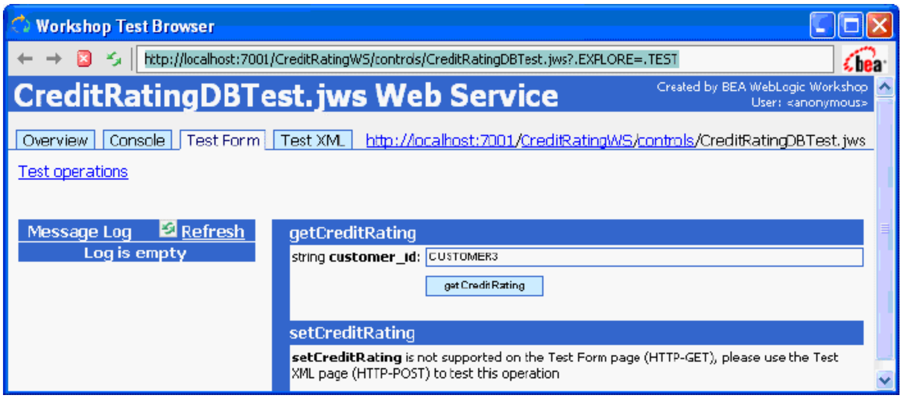
4. Test the imported Web service, by completing the following steps:

   a. Click the Start icon, or press Ctrl + F5, to open Workshop Test Browser.

   b. Enter CUSTOMER3 in the customer_id field.

   c. Click getCreditRating. The requested information displays in Workshop Test Browser.

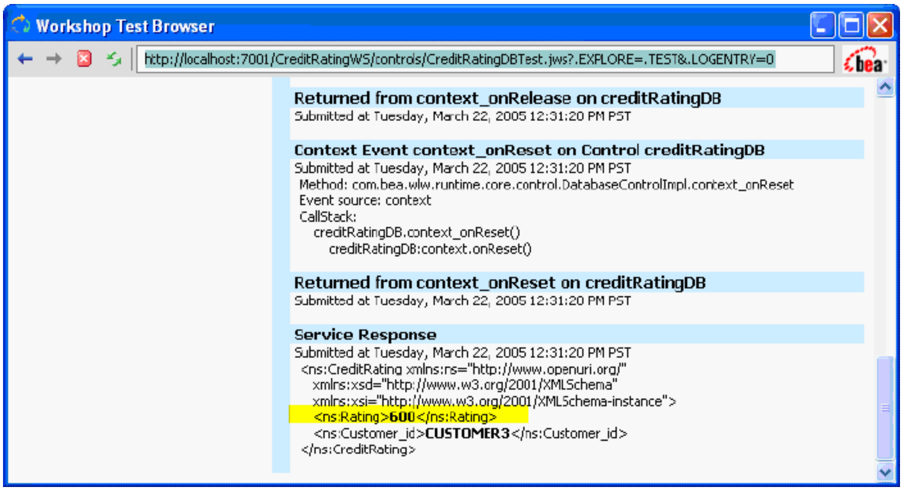
Figure 6-4 Workshop Test Browser


Workshop Test Browser
http://localhost:7001/CreditRatingWS/controls/CreditRatingDBTest.jws?.EXPLORE=.TEST
CreditRatingDBTest.jws Web Service
Created by BEA WebLogic Workshop
User: <anonymous>
Overview
Console
Test Form
Test XML
http://localhost:7001/CreditRatingWS/controls/CreditRatingDBTest.jws
Test operations
Message Log
Refresh
Log is empty
getCreditRating
string customer_id: CUSTOMER3
getCreditRating
setCreditRating

setCreditRating is not supported on the Test Form page (HTTP-GET), please use the Test XML page (HTTP-POST) to test this operation

   d. Scroll down to the Service Response section and confirm that you can retrieve credit rating information for CUSTOMER3.

Figure 6-5 Web Service Results


Workshop Test Browser
http://localhost:7001/CreditRatingWS/controls/CreditRatingDBTest.jws?.EXPLORE=.TEST&.LOGENTRY=0

```
Returned from context_onRelease on creditRatingDB
Submitted at Tuesday, March 22, 2005 12:31:20 PM PST

Context Event context_onReset on Control creditRatingDB
Submitted at Tuesday, March 22, 2005 12:31:20 PM PST
 Method: com.bea.wlw.runtime.core.control.DatabaseControlImpl.context_onReset
 Event source: context
 CallStack:
    creditRatingDB.context_onReset()
       creditRatingDB:context.onReset()

Returned from context_onReset on creditRatingDB
Submitted at Tuesday, March 22, 2005 12:31:20 PM PST

Service Response
Submitted at Tuesday, March 22, 2005 12:31:20 PM PST
 <ns:CreditRating xmlns:ns="http://www.openuri.org/"
   xmlns:xsd="http://www.w3.org/2001/XMLSchema"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
   <ns:Rating>600</ns:Rating>
   <ns:Customer_id>CUSTOMER3</ns:Customer_id>
 </ns:CreditRating>
```

6.2 Importing Web Service Metadata into a Project

WSDL is a standard XML document type for describing an associated Web service so that other software applications can interface with the Web service. Files with the `.wsdl` extension contain Web service interfaces expressed in the Web Service Description Language (WSDL).

A WSDL file contains all the information necessary for a client to invoke the methods of a Web service:

- The data types used as method parameters or return values.
- The individual method names and signatures (WSDL refers to methods as operations).
- The protocols and message formats allowed for each method.
- The URLs used to access the Web service.

Objectives

In this exercise, you will:

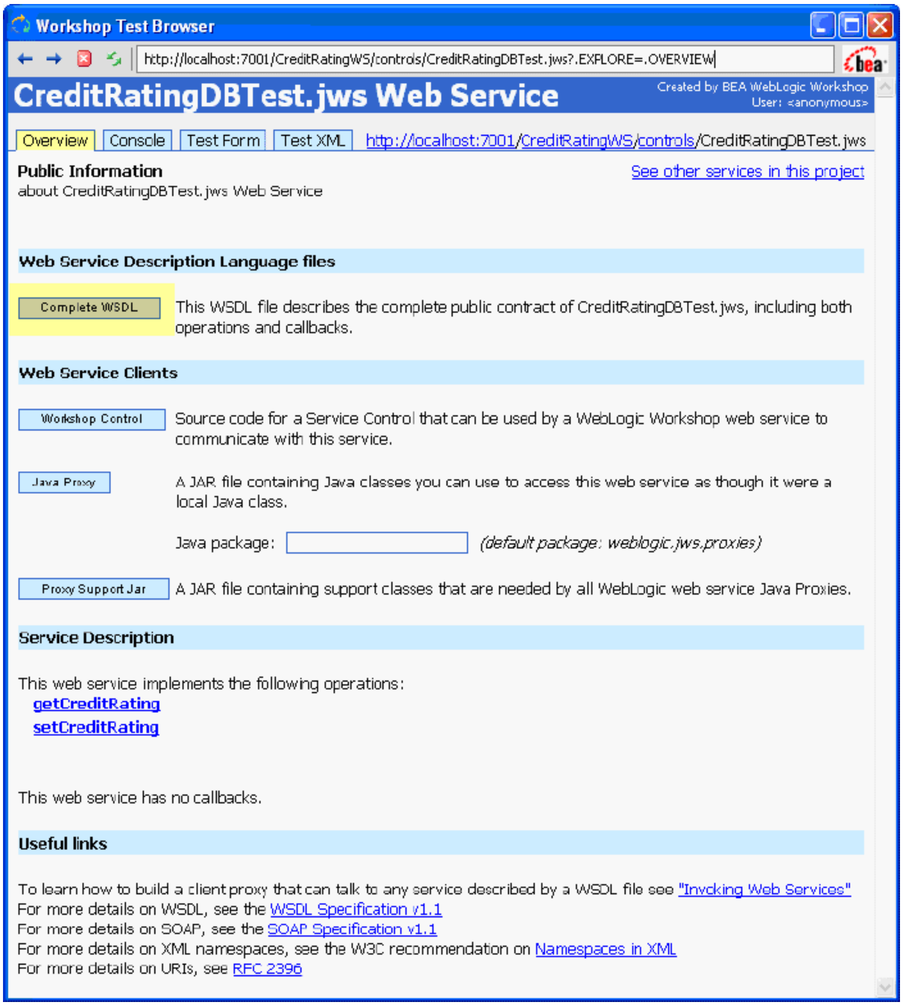
- Import the CreditRatingWS source metadata via its WSDL, into the DataServices project, thereby generating a new data service (`getCreditRatingResponse.ds`).

- Confirm that the new data service includes the `getCreditRating()` function that you tested in the previous exercise.

Instructions

1. In Workshop Test Browser, scroll to the top of the window.
2. Click the Overview tab.

Figure 6-6 Workshop Test Browser Overview



CreditRatingDBTest.jws Web Service

Created by BEA WebLogic Workshop
User: <anonymous>

http://localhost:7001/CreditRatingWS/controls/CreditRatingDBTest.jws

Public Information
about CreditRatingDBTest.jws Web Service

See other services in this project

Web Service Description Language files

Complete WSDL — This WSDL file describes the complete public contract of CreditRatingDBTest.jws, including both operations and callbacks.

Web Service Clients

Workshop Control — Source code for a Service Control that can be used by a WebLogic Workshop web service to communicate with this service.

Java Proxy — A JAR file containing Java classes you can use to access this web service as though it were a local Java class.

Java package: (default package: weblogic.jws.proxies)

Proxy Support Jar — A JAR file containing support classes that are needed by all WebLogic web service Java Proxies.

Service Description

This web service implements the following operations:
getCreditRating
setCreditRating

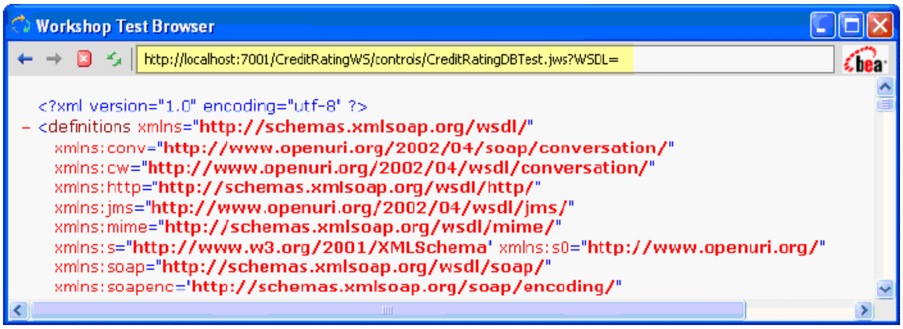
This web service has no callbacks.

Useful links

To learn how to build a client proxy that can talk to any service described by a WSDL file see "Invoking Web Services"
For more details on WSDL, see the WSDL Specification v1.1
For more details on SOAP, see the SOAP Specification v1.1
For more details on XML namespaces, see the W3C recommendation on Namespaces in XML
For more details on URIs, see RFC 2396

3. Click Complete WSDL.
4. Copy the WSDL URI, located in the Address field. The URI is typically:
   `http://localhost:7001/CreditRatingWS/controls/CreditRatingDBTest.jws?WSDL=`

Figure 6-7 WSDL URI

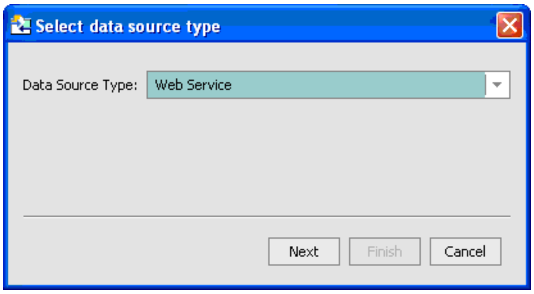


http://localhost:7001/CreditRatingWS/controls/CreditRatingDBTest.jws?WSDL=

```
<?xml version="1.0" encoding="utf-8" ?>
- <definitions xmlns="http://schemas.xmlsoap.org/wsdl/"
    xmlns:conv="http://www.openuri.org/2002/04/soap/conversation/"
    xmlns:cw="http://www.openuri.org/2002/04/wsdl/conversation/"
    xmlns:http="http://schemas.xmlsoap.org/wsdl/http/"
    xmlns:jms="http://www.openuri.org/2002/04/wsdl/jms/"
    xmlns:mime="http://schemas.xmlsoap.org/wsdl/mime/"
    xmlns:s="http://www.w3.org/2001/XMLSchema" xmlns:s0="http://www.openuri.org/"
    xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
    xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
```

5. Close Workshop Test Browser.
6. In Workshop: Close all open files (File → Close All Files).
7. Create a new folder within the DataServices project folder, and name it WebServices.
8. Import Web service source metadata into the WebServices folder, by completing the following steps:
   a. Right-click the WebServices folder.
   b. Choose Import Source Metadata.
   c. Choose Web Service from the Data Source Type drop-down list. Then click Next.

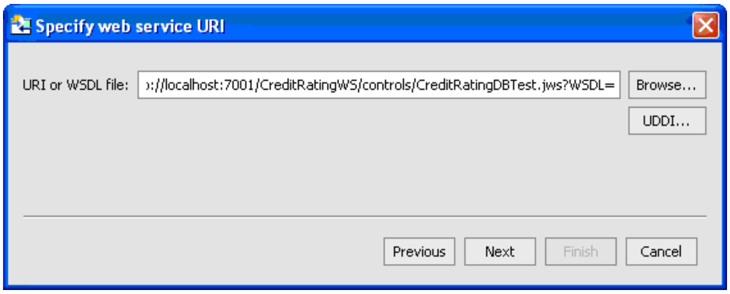
Figure 6-8 Web Service Data Source Type



Data Source Type: Web Service

   d. Paste the copied WSDL URI into the URI or WSDL File box and click Next.

Figure 6-9 Paste the URI



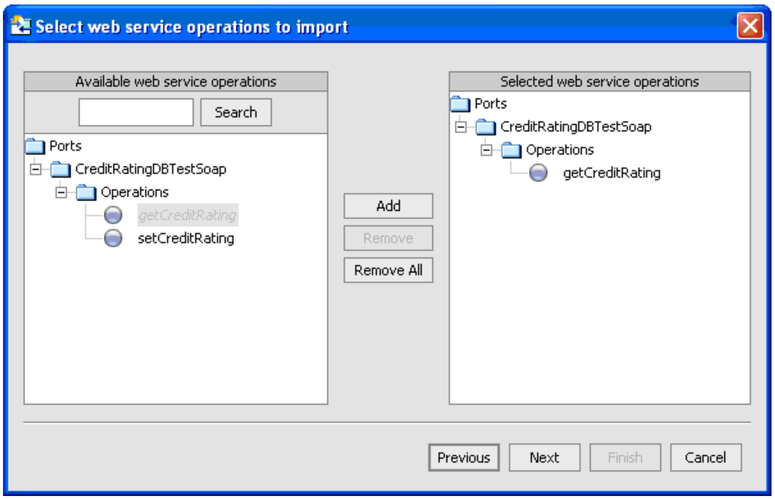
URI or WSDL file: ›://localhost:7001/CreditRatingWS/controls/CreditRatingDBTest.jws?WSDL=

e. Expand the CreditRatingDBTestSoap and Operations folders.

f. Select getCreditRating operation, and click Add to populate the Selected Web Service Operations pane.

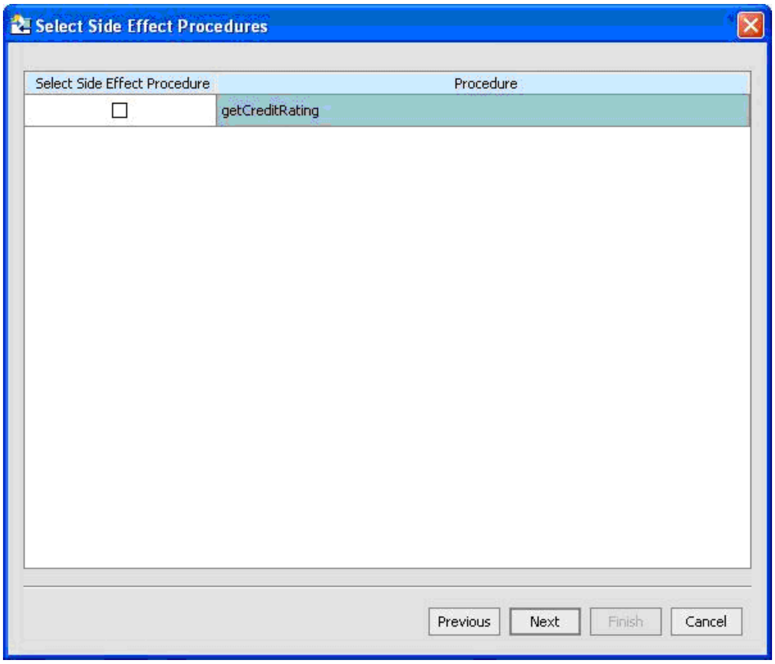
g. Click Next.

Figure 6-10 Selected Web Service Operations



h. Do not select the getCreditRating procedure as the side effect procedure in the Select Side Effect Procedures dialog box. Click Next.

Figure 6-11 Data Service Procedure Option (Unselected)

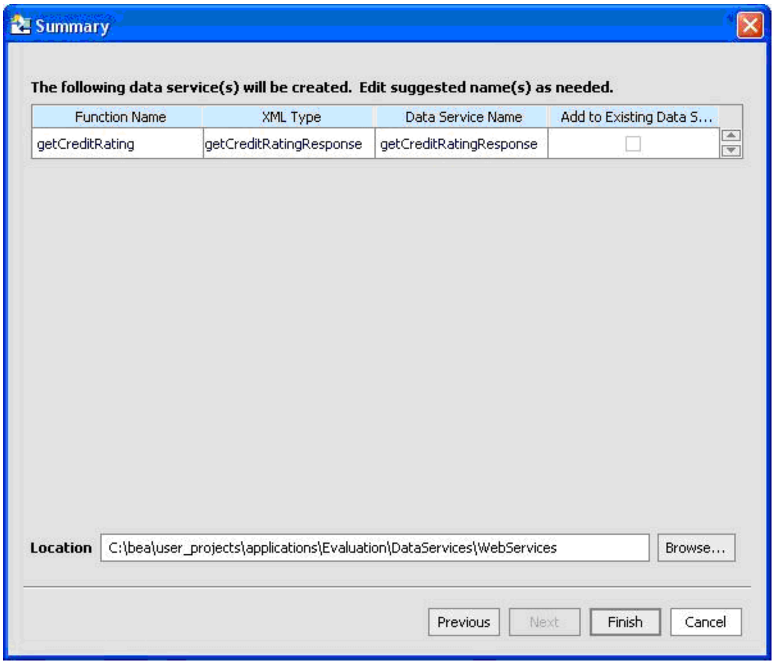


Select Side Effect Procedure	Procedure
☐	getCreditRating

i. Review the Summary information, which includes:
   - Function name.
   - *XML type*, for Web service objects whose source metadata will be imported.
   - *Name*, for each data service that will be generated from the source metadata. (Any name conflicts appear in red and must be resolved before proceeding. However, you can modify any data service name.)
   - *Add to Existing Data Service*, to add the function to an existing data service.
   - *Location*, where the generated data service(s) will reside.

j. Click Finish.

Figure 6-12 Web Services Summary



The following data service(s) will be created. Edit suggested name(s) as needed.

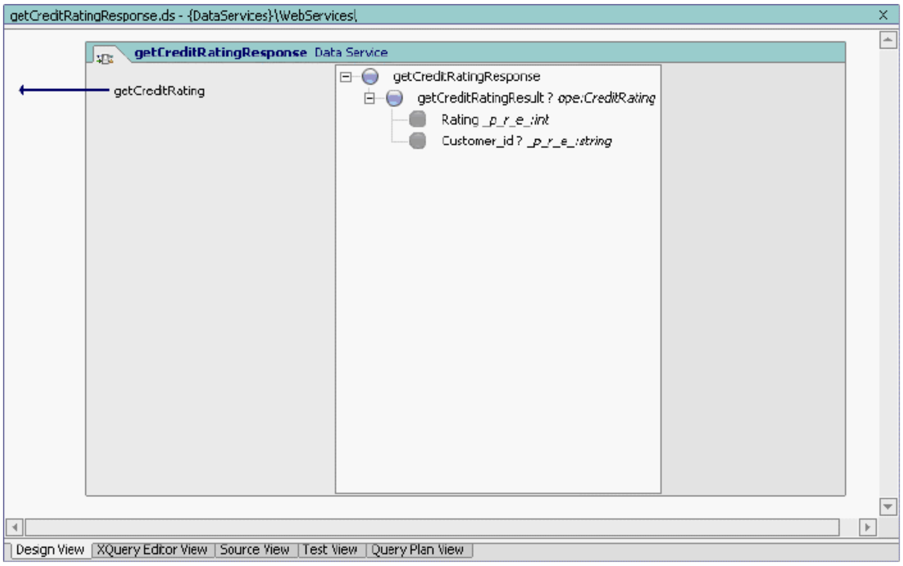
Function Name	XML Type	Data Service Name	Add to Existing Data S...
getCreditRating	getCreditRatingResponse	getCreditRatingResponse	☐

Location: C:\bea\user_projects\applications\Evaluation\DataServices\WebServices

9. Open `getCreditRatingResponse.ds` in Design View. This file is located in DataServices\WebServices.

10. Confirm that there is a function called `getCreditRating()`.

Figure 6-13 Web Service Function Added


getCreditRatingResponse.ds - {DataServices}\WebServices\
getCreditRatingResponse Data Service
getCreditRating
getCreditRatingResponse
getCreditRatingResult ? ope:CreditRating
Rating _p_r_e_:int
Customer_id ? _p_r_e_:string
Design View
XQuery Editor View
Source View
Test View
Query Plan View

6.3 Testing the Web Service via a SOAP Request

Extensible Markup Language (XML) messages provide a common language by which different applications can talk to one another over a network. Most Web services communicate via XML. A client sends an XML message containing a request to the Web service, and the Web service responds with an XML message containing the results of the operation. In most cases these XML messages are formatted according to Simple Object Access Protocol (SOAP) syntax. SOAP specifies a standard format for applications to call each other's methods and pass data to one another.

Note: Web services may communicate with XML messages that are not SOAP-formatted. The types of messages supported by a particular Web service are described in the service's WSDL file.

Objectives

In this exercise, you will:

- Use the `getCreditRating()` function and a SOAP parameter to test `getCreditRatingResponse.ds`.
- Review the results.

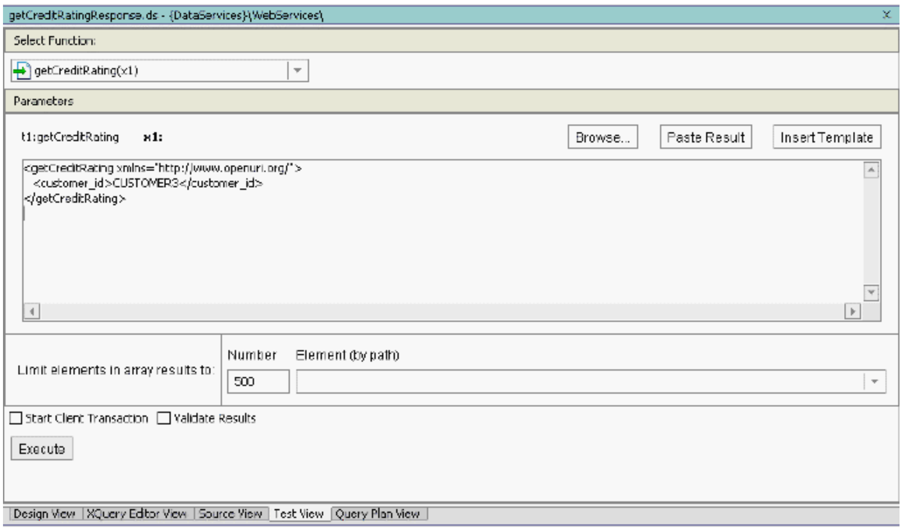
Instructions

1. Build the DataServices project.
2. Open `getCreditRatingResponse.ds` in Test View. (This file is located in DataServices\WebServices.)
3. Select `getCreditRating(x1)` from the Function drop-down list.
4. Enter the following SOAP body in the Parameter field:

```
<getCreditRating xmlns="http://www.openuri.org/">
  <customer_id>CUSTOMER3</customer_id>
</getCreditRating>
```

Note: An alternative to adding the SOAP body in the parameter field is to use a template for the input parameter by clicking Insert Template.

Figure 6-14 SOAP Parameter


getCreditRatingResponse.ds - {DataServices}\WebServices\

Select Function: getCreditRating(x1)

Parameters

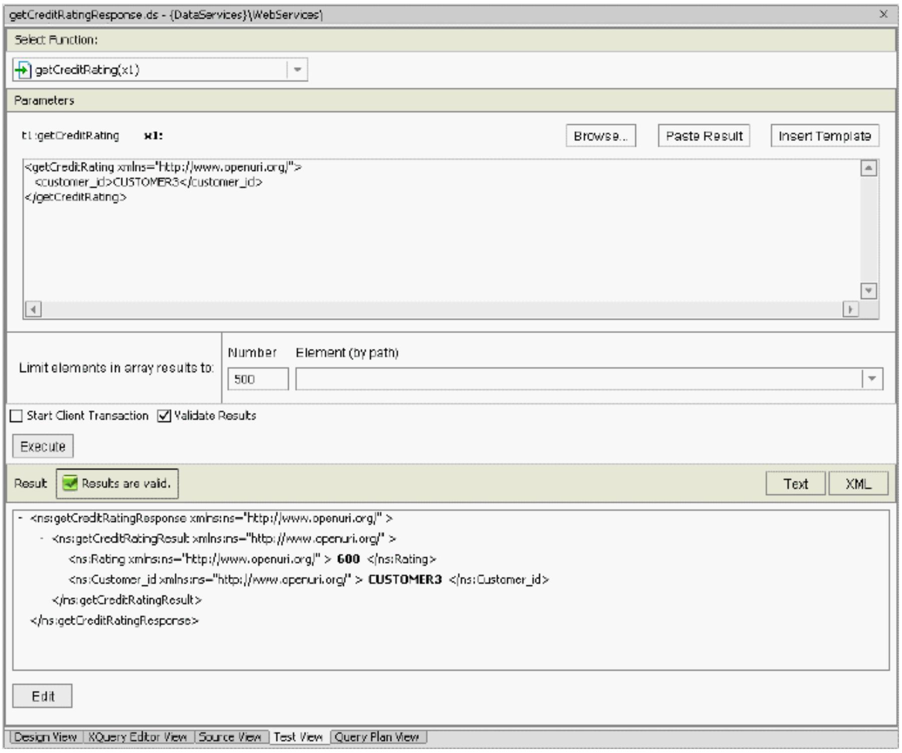
t1:getCreditRating x1:

```
<getCreditRating xmlns="http://www.openuri.org/">
  <customer_id>CUSTOMER3</customer_id>
</getCreditRating>
```

Limit elements in array results to: Number 500 Element (by path)

5. Click Execute.
6. Review the results, which should be similar to those displayed in Figure 6-15 (Rating:600, CustomerID: CUSTOMER3). Notice that only two data elements are returned: the customer ID and the credit rating for that customer.

Figure 6-15 Web Service Results



Number: 500

```
<getCreditRating xmlns="http://www.openuri.org/">
  <customer_id>CUSTOMER3</customer_id>
</getCreditRating>
```

```
- <ns:getCreditRatingResponse xmlns:ns="http://www.openuri.org/" >
   - <ns:getCreditRatingResult xmlns:ns="http://www.openuri.org/" >
        <ns:Rating xmlns:ns="http://www.openuri.org/" > 600 </ns:Rating>
        <ns:Customer_id xmlns:ns="http://www.openuri.org/" > CUSTOMER3 </ns:Customer_id>
     </ns:getCreditRatingResult>
  </ns:getCreditRatingResponse>
```

6.4 Invoking a Web Service in a Data Service

You are now ready to use the Web service to provide the data that populates the CustomerProfile logical data service.

Objectives

In this exercise, you will:

- Use the getCreditRatingResponse data service to populate the credit rating element in the CustomerProfile data service.
- Test the invocation.
- Review the results.

Instructions

1. Open `CustomerProfile.ds` file in Source View. The file is located in DataServices\CustomerManagement.

2. In the Source View, add the following namespace definitions, in addition to the ones already defined for the CustomerProfile data service:

```
declare namespace
ws1="ld:DataServices/WebServices/getCreditRatingResponse";

declare namespace ws2 = "http://www.openuri.org/";
```

Note: The “1” in “ws1” is a numeral.

3. Open the `creditRatingXQuery.txt` file, located in <beahome>\weblogic81\samples\LiquidData\EvalGuide in a text editor.

4. Copy all the code from the `creditRatingXQuery.txt` file.

5. In the CustomerProfile.ds file, expand the `getAllCustomers()` function.

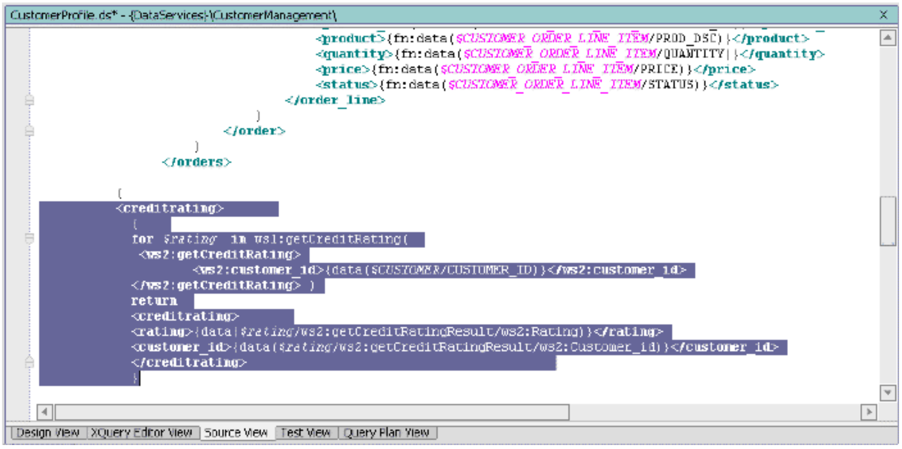
6. Insert the copied text into the section where the empty CreditRating complex element is located. The empty complex element is as follows:

```
<creditrating>

   <rating></rating/>

   <customer_id></customer_id>

</creditrating>
```

Note: The copied code replaces everything after: </orders> and before <valuation>.

7. Confirm that the <creditrating> code is as displayed in Figure 6-16.

Figure 6-16 Credit Rating Source Code



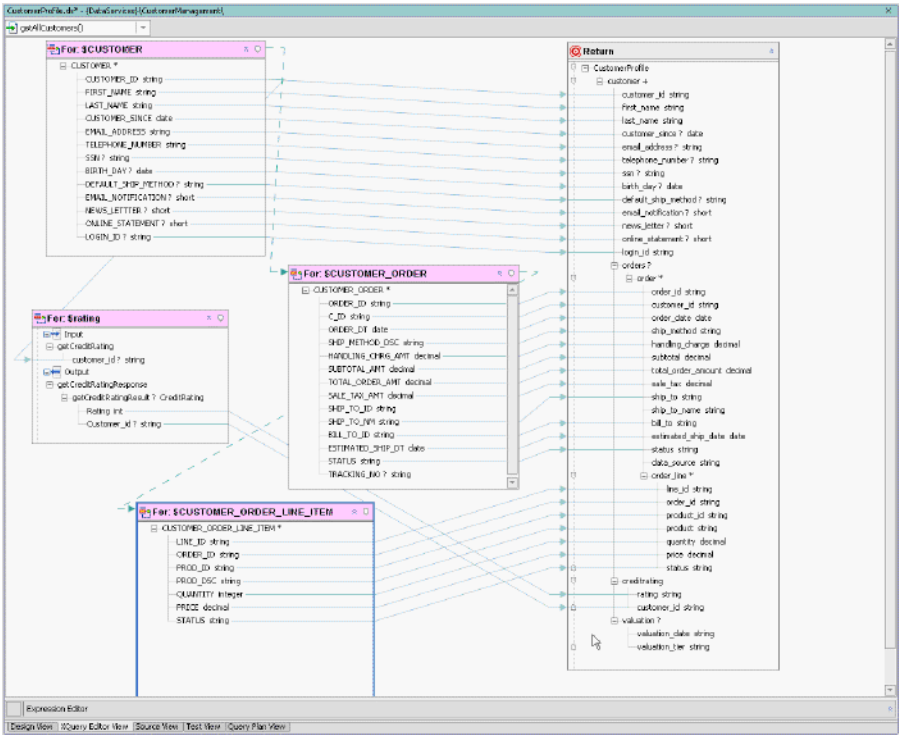
```
                                        <product>{fn:data($CUSTOMER_ORDER_LINE_ITEM/PROD_DSC)}</product>
                                        <quantity>{fn:data($CUSTOMER_ORDER_LINE_ITEM/QUANTITY)}</quantity>
                                        <price>{fn:data($CUSTOMER_ORDER_LINE_ITEM/PRICE)}</price>
                                        <status>{fn:data($CUSTOMER_ORDER_LINE_ITEM/STATUS)}</status>
                                    </order_line>
                                }
                            </order>
                        }
                    </orders>

                {
                <creditrating>
                  {
                  for $rating  in ws1:getCreditRating(
                   <ws2:getCreditRating>
                          <ws2:customer_id>{data($CUSTOMER/CUSTOMER_ID)}</ws2:customer_id>
                  </ws2:getCreditRating> )
                  return
                  <creditrating>
                  <rating>{data($rating/ws2:getCreditRatingResult/ws2:Rating)}</rating>
                  <customer_id>{data($rating/ws2:getCreditRatingResult/ws2:Customer_id)}</customer_id>
                  </creditrating>
                  }
```

8. View the results, by completing the following steps:

   a. Open `CustomerProfile.ds` in XQuery Editor View.

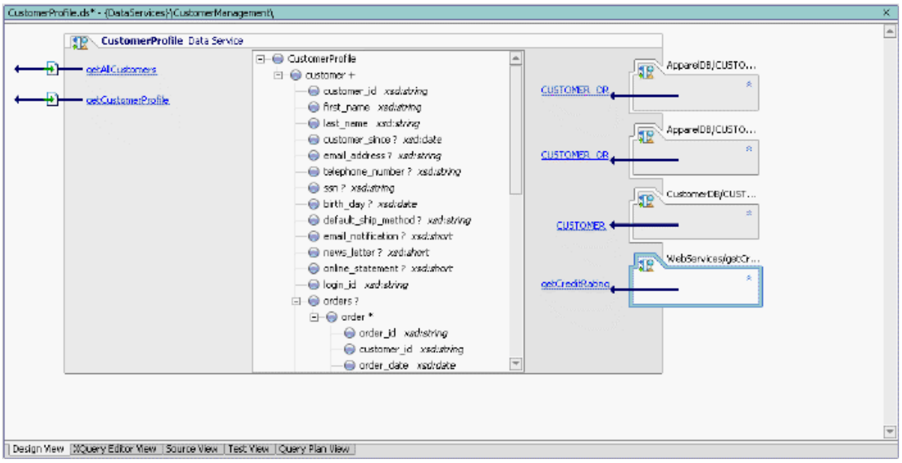
   b. Select `getAllCustomers()` from the Function dropdown list. The function should be similar to that displayed in Figure 6-17.

Figure 6-17 XQuery Editor View of a Web Service Being Invoked



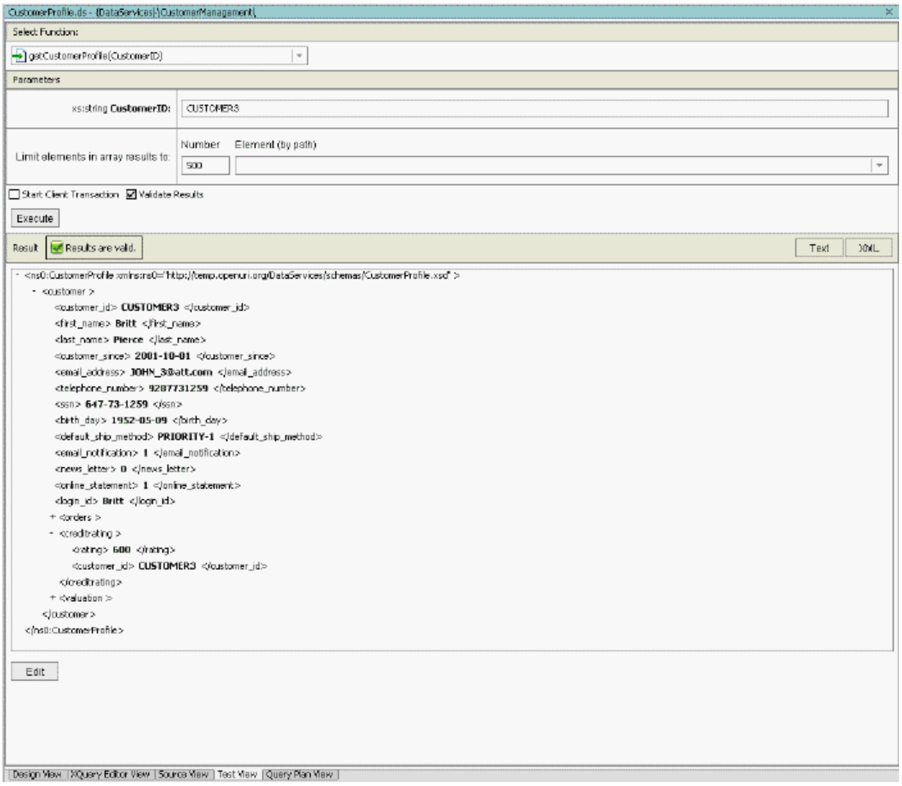
c. Open `CustomerProfile.ds` in Design View. The Web service is listed as a data source, in the right pane of the diagram.

Figure 6-18 Design View of a Web Service Invoked in a Data Service



9. Test the data service by completing the following steps:
   a. Build the DataServices project.
   b. Open `CustomerProfile.ds` in Test View.
   c. Select `getCustomerProfile(CustomerID)` from the Function drop-down list.
   d. Enter CUSTOMER3 in the xs:string CustomerID field.
   e. Click Execute.
   f. Confirm that you can retrieve the credit rating for Customer 3.

Figure 6-19 Customer Profile Data Integrated with Web Service Credit Rating Data



getCustomerProfile(CustomerID)

xs:string **CustomerID:** CUSTOMER3

Limit elements in array results to: Number 500 Element (by path)

☐ Start Client Transaction ☑ Validate Results

Results are valid.

```
- <ns0:CustomerProfile xmlns:ns0="http://temp.openuri.org/DataServices/schemas/CustomerProfile.xsd" >
  - <customer >
      <customer_id> CUSTOMER3 </customer_id>
      <first_name> Britt </first_name>
      <last_name> Pierce </last_name>
      <customer_since> 2001-10-01 </customer_since>
      <email_address> JOHN_3@att.com </email_address>
      <telephone_number> 9287731259 </telephone_number>
      <ssn> 647-73-1259 </ssn>
      <birth_day> 1952-05-09 </birth_day>
      <default_ship_method> PRIORITY-1 </default_ship_method>
      <email_notification> 1 </email_notification>
      <news_letter> 0 </news_letter>
      <online_statement> 1 </online_statement>
      <login_id> Britt </login_id>
    + <orders >
    - <creditrating >
        <rating> 600 </rating>
        <customer_id> CUSTOMER3 </customer_id>
      </creditrating>
    + <valuation >
  </customer>
</ns0:CustomerProfile>
```

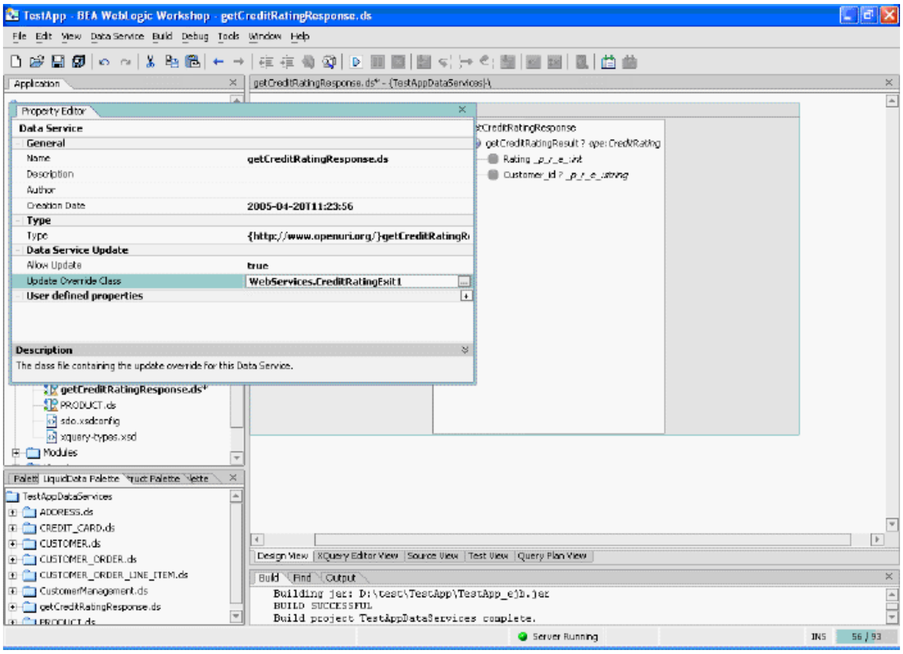
10. Import the CreditRatingExit1.java file from the EvalGuide folder:
    a. Right-click the WebServices folder.
    b. Select Import option.
    c. Navigate to <beahome>\weblogic81\samples\LiquidData\EvalGuide and select file `CreditRatingExit1.java` for import. Click Import.
    d. Build the DataServices project.
    e. Open `getCreditRatingResponse.ds` in Design View. Set the UpdateOverride Class property in the Property Editor to `WebServices.CreditRatingExit1`. (If the Property Editor is not open, you can select it using the View menu Property Editor option.)

f. Click the browser symbol in the Update Override Class field.

g. Navigate to the `DataServices.jar` -> WebServices folder.

h. Select the `CreditRatingExit1.class` file. Click open.

Figure 6-20 Selecting the Update Override Class


TestApp - BEA WebLogic Workshop - getCreditRatingResponse.ds
File Edit View Data Service Build Debug Tools Window Help
Application
getCreditRatingResponse.ds* - {TestAppDataServices}\
Property Editor
getCreditRatingResponse.ds*
PRODUCT.ds
sdo.xsdconfig
xquery-types.xsd
Modules
Palett LiquidData Palette ruct Palette ette
TestAppDataServices
ADDRESS.ds
CREDIT_CARD.ds
CUSTOMER.ds
CUSTOMER_ORDER.ds
CUSTOMER_ORDER_LINE_ITEM.ds
CustomerManagement.ds
getCreditRatingResponse.ds
PRODUCT.ds
Design View XQuery Editor View Source View Test View Query Plan View
Build Find Output
Server Running
INS
56 / 93

Data Service	
General	
Name	getCreditRatingResponse.ds
Description	
Author	
Creation Date	2005-04-20T11:23:56
Type	
Type	{http://www.openuri.org/}getCreditRatingR
Data Service Update	
Allow Update	true
Update Override Class	WebServices.CreditRatingExit1
User defined properties	

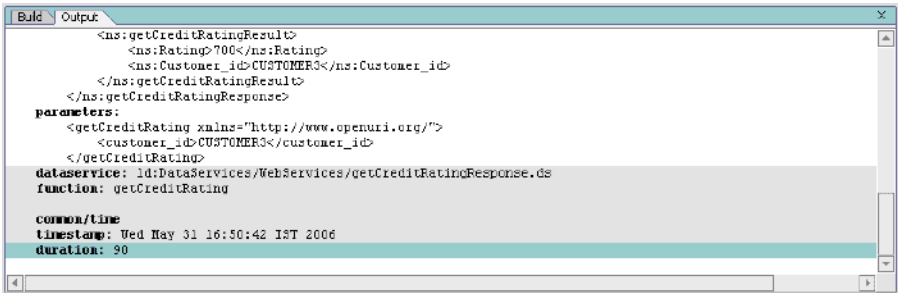
Description
The class file containing the update override for this Data Service.

```
Building jar: D:\test\TestApp\TestApp_ejb.jar
BUILD SUCCESSFUL
Build project TestAppDataServices complete.
```

11. (*Optional*) Open the Output window to view the data sources used to generate the Test View results. You should see the following statement, which indicates that data was pulled from the invoked Web service:

Note: To perform this step, you need to enable auditing in the ALDSP Console.

Figure 6-21 Viewing the Data Sources in the Output Window



```
            <ns:getCreditRatingResult>
                <ns:Rating>700</ns:Rating>
                <ns:Customer_id>CUSTOMER3</ns:Customer_id>
            </ns:getCreditRatingResult>
        </ns:getCreditRatingResponse>
    parameters:
        <getCreditRating xmlns="http://www.openuri.org/">
            <customer_id>CUSTOMER3</customer_id>
        </getCreditRating>
    dataservice: ld:DataServices/WebServices/getCreditRatingResponse.ds
    function: getCreditRating

    common/time
    timestamp: Wed May 31 16:50:42 IST 2006
    duration: 90
```

Lesson Summary

In this lesson, you learned how to:

- Import a Web service project, locate its WSDL, and use that WSDL to generate a data source.
- Test the Web service by passing a SOAP request body as a query parameter.
- Use a logical data service to invoke a Web service and retrieve data.

Consuming Data Services Using Java

After a Data Services Platform (ALDSP) application is deployed to a WebLogic Server, clients can use it to access real-time data. ALDSP supports a services-oriented approach to data access, using several technologies:

- **Mediator API**. The Java-based Mediator API instantiates ALDSP information as data objects, which are defined by the Service Data Objects (SDO) specification. SDO is a proposed standard that defines a language and architecture intended to simplify and unify the way applications handle data.
- **Data Services** Workshop Control. The Data Services Workshop control is a wizard-generated Java file that exposes a user-specified data service function to WebLogic Workshop client applications (such as page flows, portals, or Web services). You can add functions to the control from data services deployed on any WebLogic server that is accessible to the client application, whether it is on the same WebLogic Server as the client application or on a remote WebLogic Server.
- **WSDL.** WSDL-based Web services can act as wrappers for data services.
- **SQL.** The Data Services Platform JDBC driver gives SQL clients (such as reporting and database tools) and JDBC applications a traditional, database-oriented view of the data layer. To users of the JDBC driver, the set of data served by ALDSP appears as a single virtual database, with each service appearing as a table.

In this lesson, you will enable ALDSP to consume data through the SDO Mediator API.

Objectives

After completing this lesson, you will be able to:

- Use SDO in a Java application.
- Invoke a data service function using the untyped SDO Mediator API interface.
- Access data services from Java, using the typed SDO Mediator API.

Overview

SDO is a joint specification of BEA and IBM that defines a Java-based programming architecture and API for data access. A central goal of SDO is to provide client applications with a unified interface for accessing and updating data, regardless of its physical source or format.

SDO has similarities with other data access technologies, such as JDBC, Java Data Objects (JDO), and XMLBeans. However, what distinguishes SDO from other technologies is that SDO gives applications both static programming and a dynamic API for accessing data, along with a disconnected model for accessing externally persisted data. Disconnected data access means that when ALDSP gets data from a source, such as a database, it opens a connection to the source only long enough to retrieve the data. The connection is closed while the client operates on the data locally. When the client submits changes to apply to the source, the connection is reopened.

ALDSP implements the SDO specification as its client programming model. In concrete terms, this means that when a client application invokes a read function on a data service residing on a server, any data is returned as a data object. A data object is a fundamental component of the SDO programming model. It represents a unit of structured information, with static and dynamic interfaces for getting and setting its properties.

In addition to static calls, SDO, like RowSets in JDBC, has a dynamic Mediator API for accessing data through untyped calls (for example, `getString("CUSTOMER_NAME")`). An untyped Mediator API is useful if you do not know the data service to run at development time.

The Mediator API gives client applications full access to data services deployed on a WebLogic server. The application can invoke read functions, get the results as Service Data Objects, and pass changes back to the source. To use the Mediator API, a client program must first establish an initial context with the server that hosts the data services. The client can then invoke data service queries and operate on the results as Service Data Objects.

7.1 Running a Java Program Using the Untyped Mediator API

An untyped Mediator API is useful if, at development time, you do not know the data service to run.

Objectives

In this exercise, you will:

- Add a Java project to your application.
- Add the method calls necessary to use the Mediator API.
- Review the results in the Output window and a standalone Java application.

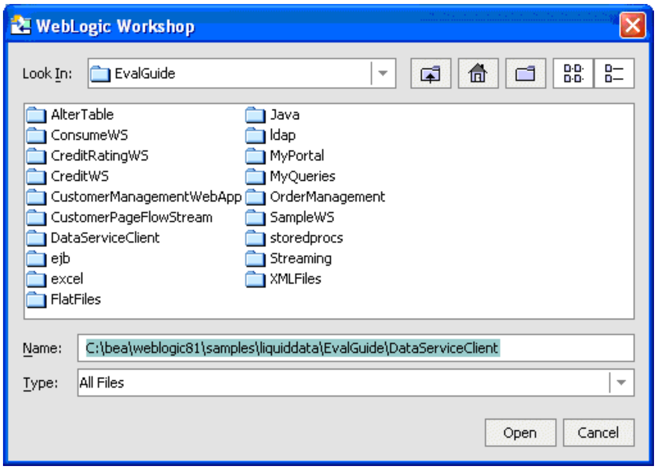
Instructions

1. Add a Java project to your application by completing the following steps:
   a. Right-click the Evaluation application folder.
   b. Select Import Project.
   c. Select Java Project.
   d. Click Browse and navigate to:

```
<beahome>\weblogic81\samples\liquiddata\EvalGuide
```

   e. Select DataServiceClient, click Open, and then click Import.

Figure 7-1 Importing Java Project



Name: C:\bea\weblogic81\samples\liquiddata\EvalGuide\DataServiceClient

The Java project is added to the application, in the DataServiceClient folder. To use the Mediator API, you need to add the method calls to instantiate the data service, invoke the `getCustomerProfile()` method and assign the return value of the function to the CustomerProfileDocument SDO/XML bean.

2. Open the `DataServiceClient.java` file, located in the DataServiceClient folder.

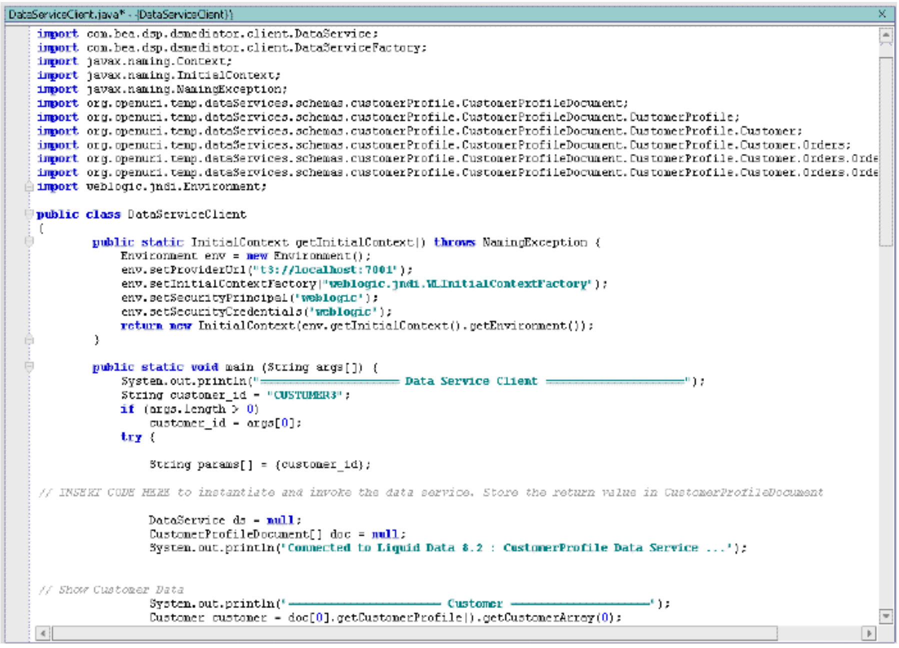
3. Insert the method calls necessary to use the Mediator API, by completing the following steps:

   a. Add the following import statements at the beginning of the file:

```
import com.bea.dsp.dsmediator.client.DataService;

import com.bea.dsp.dsmediator.client.DataServiceFactory;
```

   b. Locate the main method. You will see a declaration of the data service, a String params[ ], plus the CustomerProfileDocument variable.

Figure 7-2 Java Source Code


DataServiceClient.java* - {DataServiceClient}

```
import com.bea.dsp.dsmediator.client.DataService;
import com.bea.dsp.dsmediator.client.DataServiceFactory;
import javax.naming.Context;
import javax.naming.InitialContext;
import javax.naming.NamingException;
import org.openuri.temp.dataServices.schemas.customerProfile.CustomerProfileDocument;
import org.openuri.temp.dataServices.schemas.customerProfile.CustomerProfileDocument.CustomerProfile;
import org.openuri.temp.dataServices.schemas.customerProfile.CustomerProfileDocument.CustomerProfile.Customer;
import org.openuri.temp.dataServices.schemas.customerProfile.CustomerProfileDocument.CustomerProfile.Customer.Orders;
import org.openuri.temp.dataServices.schemas.customerProfile.CustomerProfileDocument.CustomerProfile.Customer.Orders.Orde
import org.openuri.temp.dataServices.schemas.customerProfile.CustomerProfileDocument.CustomerProfile.Customer.Orders.Orde
import weblogic.jndi.Environment;

public class DataServiceClient
{
        public static InitialContext getInitialContext() throws NamingException {
            Environment env = new Environment();
            env.setProviderUrl("t3://localhost:7001");
            env.setInitialContextFactory("weblogic.jndi.WLInitialContextFactory");
            env.setSecurityPrincipal("weblogic");
            env.setSecurityCredentials("weblogic");
            return new InitialContext(env.getInitialContext().getEnvironment());
        }

        public static void main (String args[]) {
            System.out.println("==================== Data Service Client ====================");
            String customer_id = "CUSTOMER3";
            if (args.length > 0)
                customer_id = args[0];
            try {

                String params[] = {customer_id};

// INSERT CODE HERE to instantiate and invoke the data service. Store the return value in CustomerProfileDocument

                DataService ds = null;
                CustomerProfileDocument[] doc = null;
                System.out.println("Connected to Liquid Data 8.2 : CustomerProfile Data Service ...");


// Show Customer Data
                System.out.println("-------------------- Customer --------------------");
                Customer customer = doc[0].getCustomerProfile().getCustomerArray(0);
```

c. Confirm that the String params[ ], which is an object array consisting of arguments to be passed to the function, is set as follows:

```
String params[] = {customer_id};
```

d. Construct a new data service instance, by modifying the DataService ds = null line. The Mediator API provides a class called DataServiceFactory, which can be used to construct the new data service instance. Using the newDataService method, you can pass in the initial JNDI context, the application name, and the data service name as parameters. For example:

```
DataService ds = DataServiceFactory.newDataService(

getInitialContext(),                    // Initial Context

"Evaluation",                   // Application Name

"ld:DataServices/CustomerManagement/CustomerProfile" // Data Service Name

);
```

e. Change the invocation of the data service by modifying the CustomerProfileDocument doc = null line, as shown in the following code:

```
CustomerProfileDocument[] doc = (CustomerProfileDocument[])
ds.invoke("getCustomerProfile",params);
```

f. Specify the first element of the customer profile array by changing the following code:

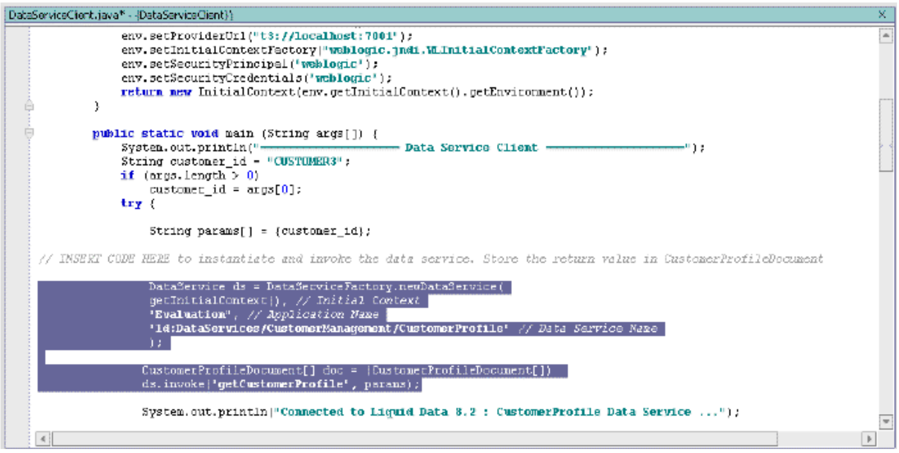
```
Customer customer = doc.getCustomerProfile().getCustomerArray(0);
```

to:

```
Customer customer = doc[0].getCustomerProfile().getCustomerArray(0);
```

g. Review the inserted code and verify that it is similar to that displayed in Figure 7-3.

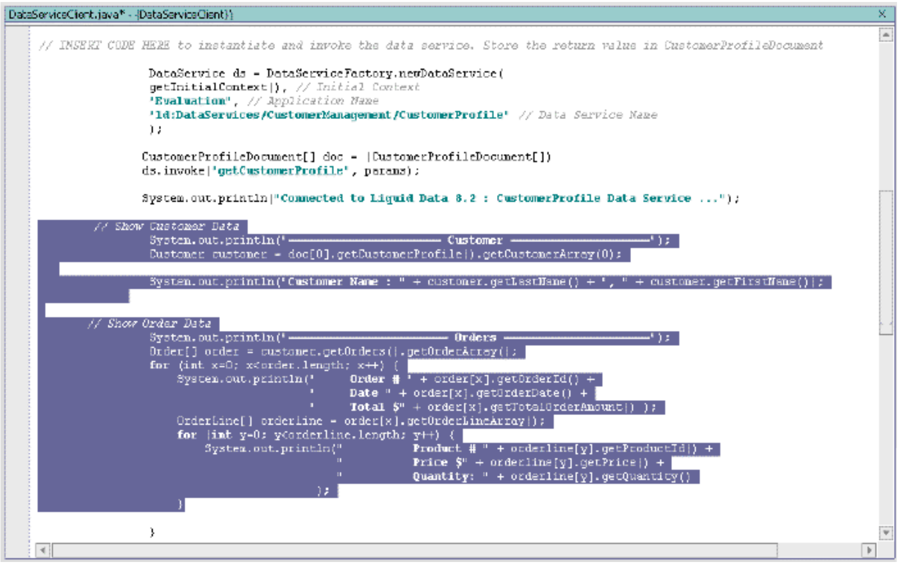
Figure 7-3 Untyped Mediator API Code Added



```
            env.setProviderUrl("t3://localhost:7001");
            env.setInitialContextFactory("weblogic.jndi.WLInitialContextFactory");
            env.setSecurityPrincipal("weblogic");
            env.setSecurityCredentials("weblogic");
            return new InitialContext(env.getInitialContext().getEnvironment());
        }

        public static void main (String args[]) {
            System.out.println("------------------- Data Service Client -------------------");
            String customer_id = "CUSTOMER3";
            if (args.length > 0)
                customer_id = args[0];
            try {

                String params[] = {customer_id};

// INSERT CODE HERE to instantiate and invoke the data service. Store the return value in CustomerProfileDocument

                DataService ds = DataServiceFactory.newDataService(
                getInitialContext(), // Initial Context
                "Evaluation", // Application Name
                "ld:DataServices/CustomerManagement/CustomerProfile" // Data Service Name
                );

               CustomerProfileDocument[] doc = (CustomerProfileDocument[])
               ds.invoke("getCustomerProfile", params);

               System.out.println("Connected to Liquid Data 8.2 : CustomerProfile Data Service ...");
```

4. Review the code included in the //Show Customer Data and //Show Order Data sections. This code will be used to retrieve customer information, all orders of that customer (order ID, order date, and total amount) and the line items of each order (product ID, price and quantity). The code should be similar to that displayed in Figure 7-4.

Figure 7-4 Customer and Order Code



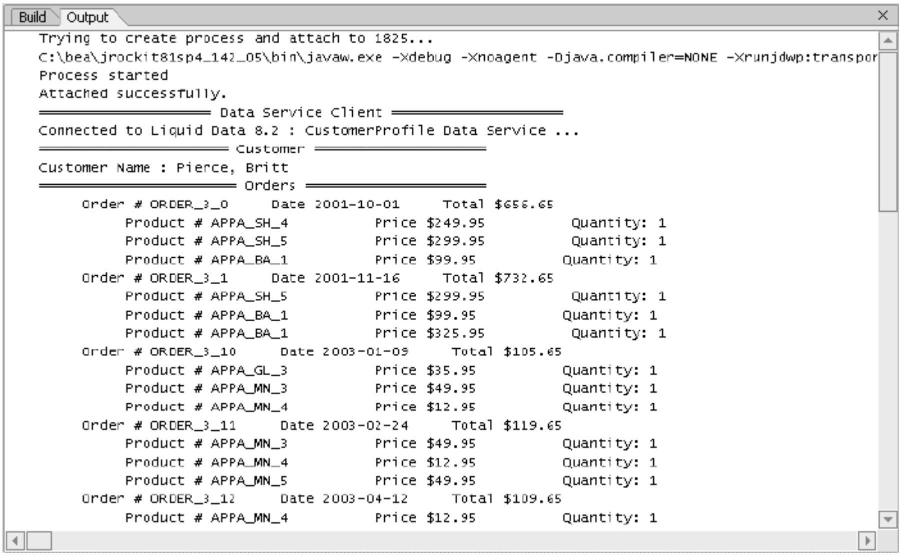
DataServiceClient.java* - {DataServiceClient}

```
// INSERT CODE HERE to instantiate and invoke the data service. Store the return value in CustomerProfileDocument

                DataService ds = DataServiceFactory.newDataService(
                getInitialContext(), // Initial Context
                "Evaluation", // Application Name
                "ld:DataServices/CustomerManagement/CustomerProfile"  // Data Service Name
                );

               CustomerProfileDocument[] doc = (CustomerProfileDocument[])
               ds.invoke("getCustomerProfile", params);

               System.out.println("Connected to Liquid Data 8.2 : CustomerProfile Data Service ...");

        // Show Customer Data
                System.out.println("———————————— Customer ————————————");
                Customer customer = doc[0].getCustomerProfile().getCustomerArray(0);

                System.out.println("Customer Name : " + customer.getLastName() + ", " + customer.getFirstName());


       // Show Order Data
                System.out.println("———————————— Orders ————————————");
                Order[] order = customer.getOrders().getOrderArray();
                for (int x=0; x<order.length; x++) {
                    System.out.println("     Order # " + order[x].getOrderId() +
                                       "     Date " + order[x].getOrderDate() +
                                       "     Total $" + order[x].getTotalOrderAmount() );
                    OrderLine[] orderline = order[x].getOrderLineArray();
                    for (int y=0; y<orderline.length; y++) {
                        System.out.println("          Product # " + orderline[y].getProductId() +
                                           "          Price $" + orderline[y].getPrice() +
                                           "          Quantity: " + orderline[y].getQuantity()
                                       );
                }

                }
```

5. Click the Start icon (or press Ctrl + F5) to compile your program (if a Confirmation message regarding debugging properties appears, then click OK). It may take a few moments to compile the program.

Note: WebLogic Server must be running. Confirm that the program returns the specified results by viewing the results in the Output window (if the Output window is not open, choose View → Windows → Output).

Figure 7-5 Results: Output Window



```
Trying to create process and attach to 1825...
C:\bea\jrockit81sp4_142_05\bin\javaw.exe -Xdebug -Xnoagent -Djava.compiler=NONE -Xrunjdwp:transpor
Process started
Attached successfully.
=================== Data Service Client ===================
Connected to Liquid Data 8.2 : CustomerProfile Data Service ...
===================== Customer ===================
Customer Name : Pierce, Britt
====================== Orders ====================
     Order # ORDER_3_0     Date 2001-10-01     Total $656.65
          Product # APPA_SH_4          Price $249.95          Quantity: 1
          Product # APPA_SH_5          Price $299.95          Quantity: 1
          Product # APPA_BA_1          Price $99.95          Quantity: 1
     Order # ORDER_3_1     Date 2001-11-16     Total $732.65
          Product # APPA_SH_5          Price $299.95          Quantity: 1
          Product # APPA_BA_1          Price $99.95          Quantity: 1
          Product # APPA_BA_1          Price $325.95          Quantity: 1
     Order # ORDER_3_10     Date 2003-01-09     Total $105.65
          Product # APPA_GL_3          Price $35.95          Quantity: 1
          Product # APPA_MN_3          Price $49.95          Quantity: 1
          Product # APPA_MN_4          Price $12.95          Quantity: 1
     Order # ORDER_3_11     Date 2003-02-24     Total $119.65
          Product # APPA_MN_3          Price $49.95          Quantity: 1
          Product # APPA_MN_4          Price $12.95          Quantity: 1
          Product # APPA_MN_5          Price $49.95          Quantity: 1
     Order # ORDER_3_12     Date 2003-04-12     Total $109.65
          Product # APPA_MN_4          Price $12.95          Quantity: 1
```

6. (Optional) View the results in a standalone Java environment of your choice.

Note: To use the Mediator API outside of WebLogic Workshop, you need to add the following files to your classpath:

- WebLogic Libraries:

`%\bea\weblogic81\server\lib\weblogic.jar`

- XML Bean:

`%\bea\weblogic81\server\lib\xbean.jar`

- CustomerProfile classes:

`%\bea\user_projects\applications\Evaluation\APP-INF\lib\DataServices.jar`

- ALDSP Server Libraries:

`%\bea\weblogic81\liquiddata\lib\ld-server-core.jar`

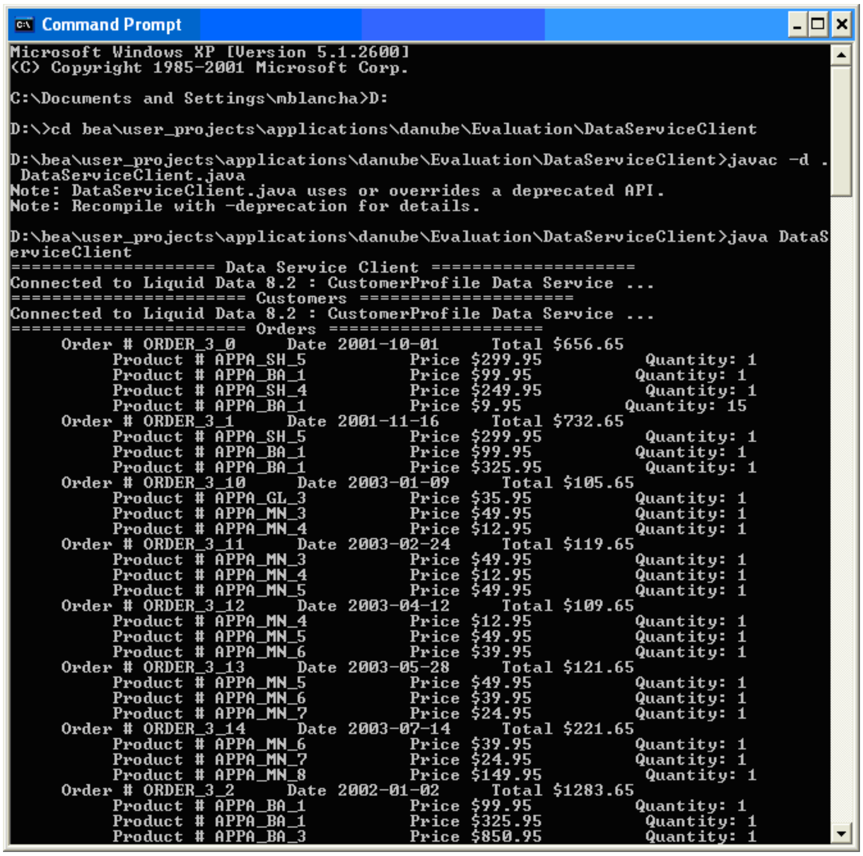
- ALDSP Client Libraries (including Mediator API):

`%\bea\weblogic81\liquiddata\lib\ld-client.jar`

- Service Data Object:

```
%\bea\weblogic81\liquiddata\lib\wlsdo.jar
```

Figure 7-6 Results: Standalone Java Environment



```
Microsoft Windows XP [Version 5.1.2600]
(C) Copyright 1985-2001 Microsoft Corp.

C:\Documents and Settings\mblancha>D:

D:\>cd bea\user_projects\applications\danube\Evaluation\DataServiceClient

D:\bea\user_projects\applications\danube\Evaluation\DataServiceClient>javac -d .
 DataServiceClient.java
Note: DataServiceClient.java uses or overrides a deprecated API.
Note: Recompile with -deprecation for details.

D:\bea\user_projects\applications\danube\Evaluation\DataServiceClient>java DataS
erviceClient
==================== Data Service Client ====================
Connected to Liquid Data 8.2 : CustomerProfile Data Service ...
======================= Customers =====================
Connected to Liquid Data 8.2 : CustomerProfile Data Service ...
======================= Orders =====================
     Order # ORDER_3_0      Date 2001-10-01      Total $656.65
          Product # APPA_SH_5          Price $299.95          Quantity: 1
          Product # APPA_BA_1          Price $99.95          Quantity: 1
          Product # APPA_SH_4          Price $249.95          Quantity: 1
          Product # APPA_BA_1          Price $9.95          Quantity: 15
     Order # ORDER_3_1      Date 2001-11-16      Total $732.65
          Product # APPA_SH_5          Price $299.95          Quantity: 1
          Product # APPA_BA_1          Price $99.95          Quantity: 1
          Product # APPA_BA_1          Price $325.95          Quantity: 1
     Order # ORDER_3_10      Date 2003-01-09      Total $105.65
          Product # APPA_GL_3          Price $35.95          Quantity: 1
          Product # APPA_MN_3          Price $49.95          Quantity: 1
          Product # APPA_MN_4          Price $12.95          Quantity: 1
     Order # ORDER_3_11      Date 2003-02-24      Total $119.65
          Product # APPA_MN_3          Price $49.95          Quantity: 1
          Product # APPA_MN_4          Price $12.95          Quantity: 1
          Product # APPA_MN_5          Price $49.95          Quantity: 1
     Order # ORDER_3_12      Date 2003-04-12      Total $109.65
          Product # APPA_MN_4          Price $12.95          Quantity: 1
          Product # APPA_MN_5          Price $49.95          Quantity: 1
          Product # APPA_MN_6          Price $39.95          Quantity: 1
     Order # ORDER_3_13      Date 2003-05-28      Total $121.65
          Product # APPA_MN_5          Price $49.95          Quantity: 1
          Product # APPA_MN_6          Price $39.95          Quantity: 1
          Product # APPA_MN_7          Price $24.95          Quantity: 1
     Order # ORDER_3_14      Date 2003-07-14      Total $221.65
          Product # APPA_MN_6          Price $39.95          Quantity: 1
          Product # APPA_MN_7          Price $24.95          Quantity: 1
          Product # APPA_MN_8          Price $149.95          Quantity: 1
     Order # ORDER_3_2      Date 2002-01-02      Total $1283.65
          Product # APPA_BA_1          Price $99.95          Quantity: 1
          Product # APPA_BA_1          Price $325.95          Quantity: 1
          Product # APPA_BA_3          Price $850.95          Quantity: 1
```

7.2 Running a Java Program Using the Typed Mediator API

With the typed mediator interface, you instantiate a typed data service proxy in the client, instead of using the generic data service interface. The typed data service interface may be easier to program and it improves code readability.

In this exercise, you will access data services from a Java client, using the typed SDO Mediator API. You will be provided with a generated API for your data service, which lets you directly invoke the actual functions as methods (for example, `ds.getCustomerProfile(customer_id)`).

Objectives

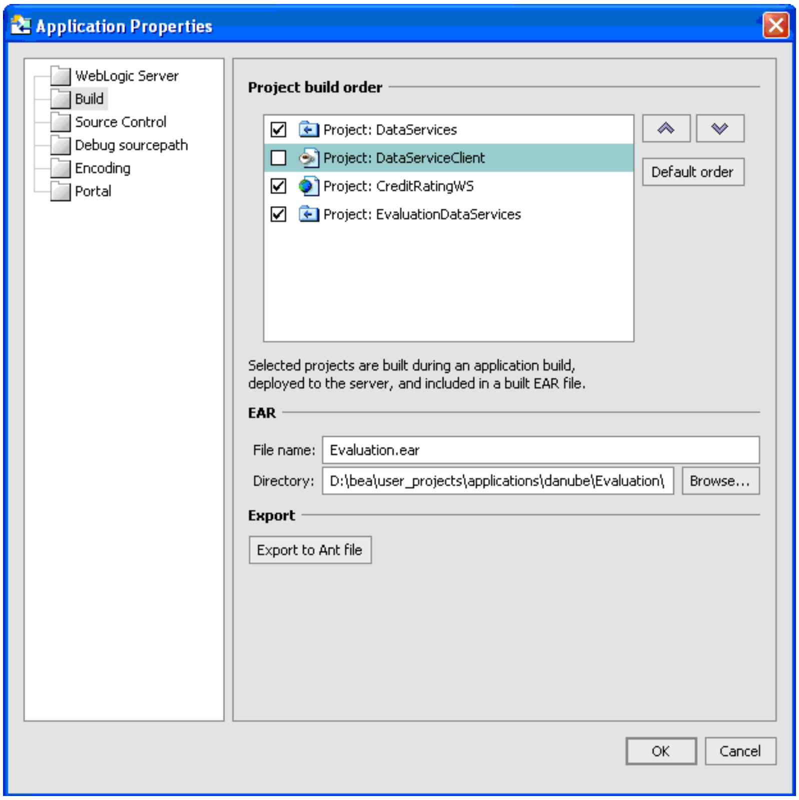
In this exercise, you will:

- Build your application as an EAR file.
- Build the SDO mediator client.
- Add the SDO mediator client's generated JAR file to your libraries folder.
- Construct a DataServices instance and invoke the data service.
- View the results in the Output window.
- View the results in a standalone Java application.

Instructions

1. Build your application as an EAR file by completing the following steps:
   a. Choose Tools → Application Properties and click Build.
   b. In the Project build order section, place DataServices as the first project.
   c. Clear the Project: DataServiceClient checkbox, because this is not required for the EAR file.
   d. Click OK.

Figure 7-7 Project Build Order



Project build order

- [✓] Project: DataServices
- [ ] Project: DataServiceClient
- [✓] Project: CreditRatingWS
- [✓] Project: EvaluationDataServices

Selected projects are built during an application build, deployed to the server, and included in a built EAR file.

EAR

File name: Evaluation.ear

Directory: D:\bea\user_projects\applications\danube\Evaluation\

Export

2. Build the SDO Mediator Client, by completing the following steps:

   a. Right-click the Evaluation application and select Build Application from the pop-up menu.

   b. Right-click the Evaluation application again and select Build SDO Mediator Client. A message displays notifying you that an EAR file will be created.

   c. Click Yes when asked whether you want to build an EAR file.

Note: This confirmation box appears only the first time you build the SDO Mediator Client. However, to ensure that the latest EAR file is used while building the SDO Mediator Client, you must build the EAR before you build the SDO Mediator Client.

   d. Confirm that you see the following text in the Build window (if not open, choose View → Windows → Build):

```
Generating SDO client API jar...

clean:

de-ear:

build:

[delete] Deleting:
C:\bea\user_projects\applications\Evaluation\Evaluation-ld-client.jar

[mkdir] Created dir: C:\Documents and Settings\jsmith\Local
Settings\Temp\wlw-temp-53911\sdo_compile42918\client\src

[java] May 2, 2006 6:41:26 PM com.bea.ld.context.MetadataContext
getRepositoryRoot

[java] INFO: 30 (ms)

[java] May 2, 2006 6:41:27 PM
com.bea.ld.wrappers.ws.JAXRPCWebserviceAdapter <clinit>

[java] WARNING: Unable to instantiate ServiceFactory. Please ensure that
javax.xml.rpc.ServiceFactory property has been properly set.

[mkdir] Created dir: C:\Documents and Settings\jsmith\Local
Settings\Temp\wlw-temp-53911\sdo_compile42918\client\classes

[javac] Compiling 12 source files to C:\Documents and
Settings\jsmith\Local
Settings\Temp\wlw-temp-53911\sdo_compile42918\client\classes

[jar] Updating jar:
C:\bea\user_projects\applications\Evaluation\Evaluation-ld-client.jar

all:

Importing SDO client API jar into application...

SDO client API jar available as
C:\bea\user_projects\applications\Evaluation\Evaluation-ld-client.jar
```

Note: The drive information may be different for your application.

3. Construct a new data service instance and invoke the data service, by completing the following steps:

   a. Open the `DataServiceClient.java` file (if it is not already open).

   b. Replace the declaration of the DataService and CustomerProfileDocument objects with the following (modified code is displayed in boldface type):

```
CustomerProfile ds = CustomerProfile.getInstance(

getInitialContext(), // Initial Context
"Evaluation" // Application Name
);
CustomerProfileDocument doc = ds.getCustomerProfile(customer_id);
```

Note: In the case of typed mediator APIs, you specify whether you are retrieving a single object or an array based on the data service function declaration. In the preceding example, to retrieve a single object in the output, the doc object is used instead of doc[0].

c. Click Alt + Enter and select dataservices.customermanagement.CustomerProfile. This creates an import statement at the beginning of the file for the specified data service.

d. Edit getInitialContext () to suit your environment. Typically no changes are needed when working through the tutorial on your local computer.

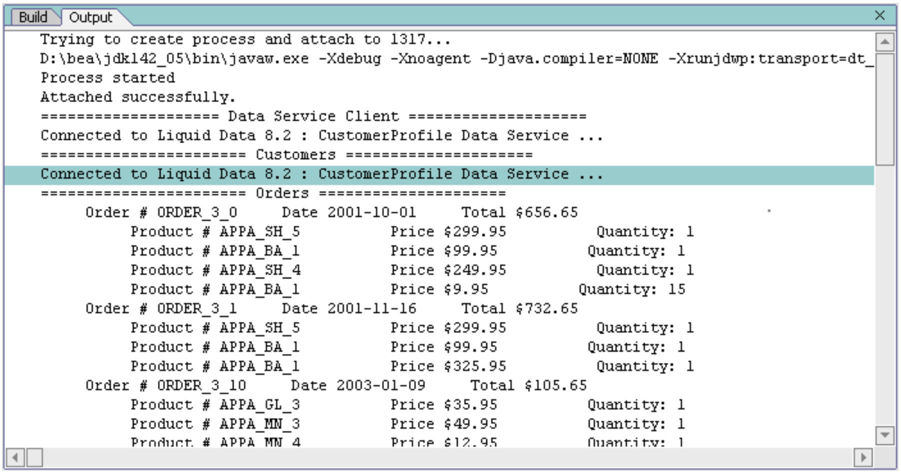
4. View the results in the Output window, by completing the following steps:

   a. Click the Start icon (or press Ctrl + F5) to compile your program.

   b. Click OK if a confirmation message asking if you would like to run DataServiceClient.

   c. Confirm that the program return the specified results by viewing the results in the Output window (if not open, choose View → Windows → Output).

Figure 7-8 Results -- Output Window

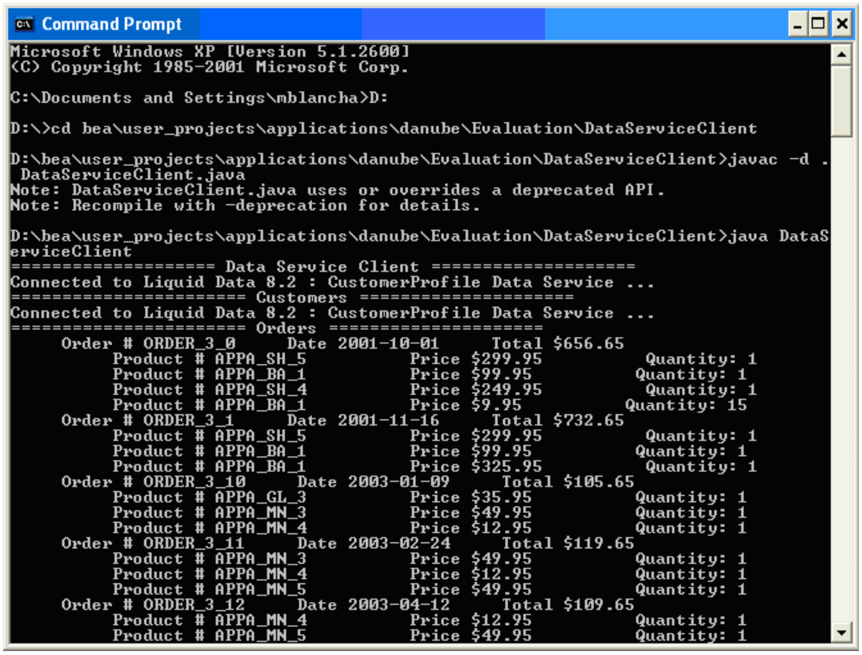


```
Trying to create process and attach to 1317...
D:\bea\jdk142_05\bin\javaw.exe -Xdebug -Xnoagent -Djava.compiler=NONE -Xrunjdwp:transport=dt_
Process started
Attached successfully.
==================== Data Service Client ====================
Connected to Liquid Data 8.2 : CustomerProfile Data Service ...
======================= Customers ====================
Connected to Liquid Data 8.2 : CustomerProfile Data Service ...
======================= Orders ====================
     Order # ORDER_3_0     Date 2001-10-01     Total $656.65
          Product # APPA_SH_5          Price $299.95          Quantity: 1
          Product # APPA_BA_1          Price $99.95          Quantity: 1
          Product # APPA_SH_4          Price $249.95          Quantity: 1
          Product # APPA_BA_1          Price $9.95          Quantity: 15
     Order # ORDER_3_1     Date 2001-11-16     Total $732.65
          Product # APPA_SH_5          Price $299.95          Quantity: 1
          Product # APPA_BA_1          Price $99.95          Quantity: 1
          Product # APPA_BA_1          Price $325.95          Quantity: 1
     Order # ORDER_3_10     Date 2003-01-09     Total $105.65
          Product # APPA_GL_3          Price $35.95          Quantity: 1
          Product # APPA_MN_3          Price $49.95          Quantity: 1
          Product # APPA_MN_4          Price $12.95          Quantity: 1
```

5. (*Optional*) Run your program in a standalone Java application to list customer orders. Note that you must add the generated file (the typed data-service proxy,

Evaluation-ld-client.jar) to the classpath, along with the other libraries listed for Excercise 7.1 Running a Java Program Using the Untyped Mediator API, (optional) step 7.

Figure 7-9 Results-- Standalone Java Application



```
Microsoft Windows XP [Version 5.1.2600]
(C) Copyright 1985-2001 Microsoft Corp.

C:\Documents and Settings\mblancha>D:

D:\>cd bea\user_projects\applications\danube\Evaluation\DataServiceClient

D:\bea\user_projects\applications\danube\Evaluation\DataServiceClient>javac -d .
 DataServiceClient.java
Note: DataServiceClient.java uses or overrides a deprecated API.
Note: Recompile with -deprecation for details.

D:\bea\user_projects\applications\danube\Evaluation\DataServiceClient>java DataS
erviceClient
==================== Data Service Client ====================
Connected to Liquid Data 8.2 : CustomerProfile Data Service ...
======================= Customers =====================
Connected to Liquid Data 8.2 : CustomerProfile Data Service ...
======================= Orders ====================
     Order # ORDER_3_0      Date 2001-10-01     Total $656.65
          Product # APPA_SH_5            Price $299.95          Quantity: 1
          Product # APPA_BA_1            Price $99.95          Quantity: 1
          Product # APPA_SH_4            Price $249.95          Quantity: 1
          Product # APPA_BA_1            Price $9.95          Quantity: 15
     Order # ORDER_3_1      Date 2001-11-16     Total $732.65
          Product # APPA_SH_5            Price $299.95          Quantity: 1
          Product # APPA_BA_1            Price $99.95          Quantity: 1
          Product # APPA_BA_1            Price $325.95          Quantity: 1
     Order # ORDER_3_10      Date 2003-01-09     Total $105.65
          Product # APPA_GL_3            Price $35.95          Quantity: 1
          Product # APPA_MN_3            Price $49.95          Quantity: 1
          Product # APPA_MN_4            Price $12.95          Quantity: 1
     Order # ORDER_3_11      Date 2003-02-24     Total $119.65
          Product # APPA_MN_3            Price $49.95          Quantity: 1
          Product # APPA_MN_4            Price $12.95          Quantity: 1
          Product # APPA_MN_5            Price $49.95          Quantity: 1
     Order # ORDER_3_12      Date 2003-04-12     Total $109.65
          Product # APPA_MN_4            Price $12.95          Quantity: 1
          Product # APPA_MN_5            Price $49.95          Quantity: 1
```

7.3 Resetting the Mediator API

After Excercise 7.2 Running a Java Program Using the Typed Mediator API, you must remove the Evaluation_ld-client.jar file from your Libraries folder because this JAR file will create inconsistencies in future lessons. You must also revert the method calls to use the Untyped Mediator API.

Objectives

In this exercise, you will:

- Remove the Evaluation_ld-client.jar file from the Libraries folder.
- Revert the method calls to use the untyped Mediator API.

Instructions

1. Delete the `Evaluation-ld-client.jar` file by completing the following steps:
   a. Expand the Libraries folder.
   b. Right-click the `Evaluation-ld-client.jar` file.
   c. Choose Delete from the pop-up menu.
   d. Click Yes, when the confirmation message displays.
2. Revert the method calls to use the untyped mediator API, by completing the following steps:
   a. Open the `DataServiceClient.java` file.
   b. Replace the declaration of the DataService and CustomerProfileDocument objects with the following (modified code is displayed in bold):

```
DataService ds = DataServiceFactory.newDataService(
getInitialContext(),                // Initial Context
"Evaluation",                 // Application Name
"ld:DataServices/CustomerManagement/CustomerProfile" // Data Service
Name
);
CustomerProfileDocument[] doc = (CustomerProfileDocument[])
ds.invoke("getCustomerProfile", params);
System.out.println("Connected to DSP 2.x : CustomerProfile Data
Service...");
```

   Note: If your application name is different from Evaluation, locate “Evaluation” in the `newDataService()` call and rename it to reflect the name of your application.

   c. Remove the import CustomerProfile statement.
   d. Save your work.

Lesson Summary

In this lesson, you learned how to:

- Set the classpath environment to use the SDO Mediator API.

- Use the untyped and typed SDO Mediator API to access data services from Java.
- Generate the specific client-side Mediator API for your data service.

Consuming Data Services using Data Service Controls

A Data Service control provides WebLogic Workshop applications with easy access to data service functions.

Objectives

After completing this lesson, you will be able to:

- Install the Data Service Control in your application.
- Create a Java page flow (`.jpf`) Web application file, using WebLogic Workshop.

Overview

A convenient way to quickly access ALDSP from a WebLogic Workshop application, such as page flows, process definitions, portals, or Web services, is through the Data Service control.

The Data Service control is a wizard-generated Java file that exposes to WebLogic Workshop client applications only those data service function that you choose. You can add functions to a control from data services deployed on any WebLogic Server that is accessible to the client application, whether it is on the same WebLogic Server as the client application or on a remote WebLogic Server.

If accessing data services on a remote server, information regarding the information that the service functions return (in the form of XML schema files) are first downloaded from the remote server into the current application. The schema files are placed in a schema project named after the remote application. The directory structure within the project mirrors the directory structure of the remote server.

When you create a Data Service control, WebLogic Workshop generates interface files for the target schemas associated with the queries and then a Java Control Extension (`.jcx`) file. The `.jcx` file contains the methods included from the data services when the control was created and a commented method that, when uncommented, allows you to pass any XQuery statement to the server in the form of an ad-hoc query.

8.1 Installing a Data Service Control

Data Service controls let you easily access data from page flows, process definitions, portals, or Web services.

Objectives

In this exercise, you will:

- Import a Web project that will be used to demonstrate Data Service control capabilities.
- Install a Data Service control.

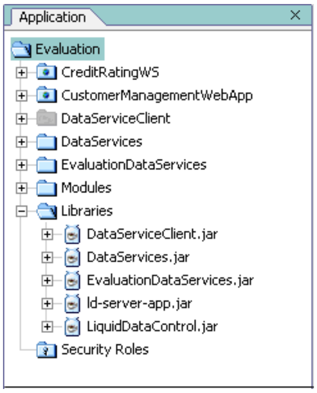
Instructions

1. Right-click the Evaluation application folder.
2. Choose Import Project.
3. Choose Web Project.
4. Navigate to <beahome>\weblogic81\samples\LiquidData\EvalGuide
5. Select the CustomerManagementWebApp project and click Open.
6. Click Import, and then click Yes when asked whether you want to install project files.
7. Right-click the Evaluation application folder.
8. Choose Install → Controls → Data Service.

Note: The Data Service option will not display if you previously installed a Data Service control.

9. Expand the Libraries folder and confirm that the `LiquidDataControl.jar` file is installed.

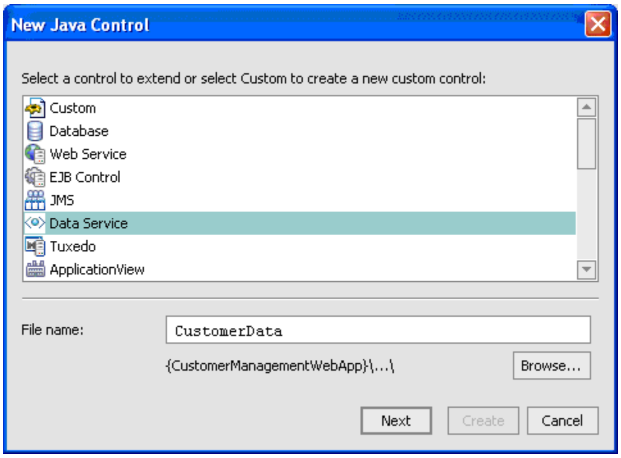
Figure 8-1 Data Service Control



8.2 Defining the Data Service Control

1. Create a new folder in the CustomerManagementWebApp Web project, and name it controls.
2. Define a new Java control as a Data Service control by completing the following steps:
   a. Right-click the controls folder.
   b. Choose New $\rightarrow$ Java Control.
   c. Select Data Service.
   d. Enter CustomerData in the File name field.
   e. Click Next.

Figure 8-2 Creating a New Java Control



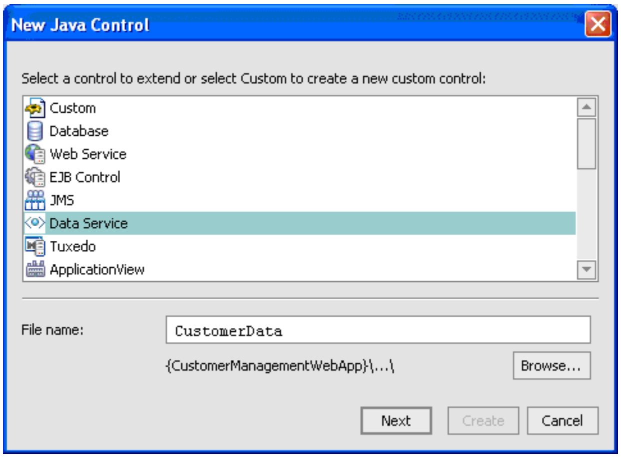
File name: CustomerData

{CustomerManagementWebApp}\...\

f. In the New Java Control – Data Service dialog box, click Create.

Note: Do not change any default settings.

Figure 8-3 Creating a New Data Service Control



File name: CustomerData

{CustomerManagementWebApp}\...\

g. In the Select Data Service Functions box, expand the CustomerManagement and then the `CustomerProfile.ds` folders.

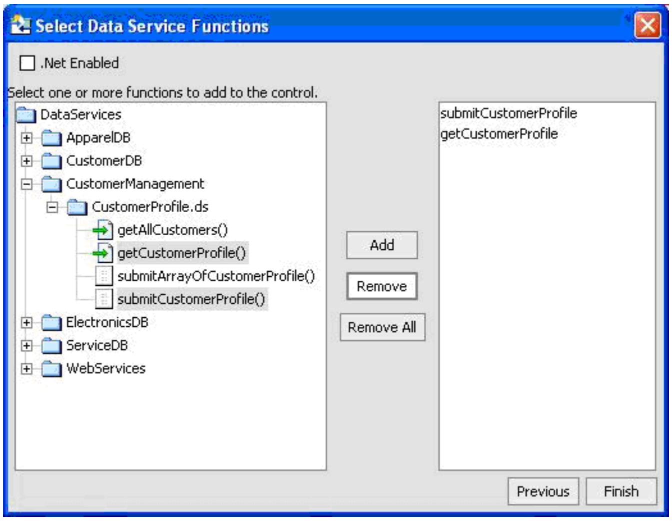
h. Select `getCustomerProfile()`.

i. Press Ctrl.

j. Select `submitCustomerProfile()`.

k. Click Add and then click Finish.

Figure 8-4 Selecting Functions for the Data Service Control



It will take a few moments for the project to compile. After compilation, you should see a Java-based Data Service Control called `CustomerData.jcx`, with the following signatures:

- `getCustomerProfile()` is a data service read function.
- `submitCustomerProfile()` is a submit function for all the changes (inserts, updates, and deletes) done to the customer profile and persisting the data to the data sources involved.

Note: You can use the data service control that you define as any WebLogic Workshop control in a workflow, a JPF, or a portal.

3. Open the `CustomerData.jcx` file in Source View. This file is located in CustomerManagementWebApp\controls.

4. Add an import statement for the filterXquery class:

```
import com.bea.ld.filter.FilterXQuery;
```

5. Select and copy the comments and definition for the `getCustomerProfile()` function. It looks like this:

```
    /**
     *
     * @jc:XDS
functionURI="ld:DataServices/CustomerManagement/CustomerProfile"
functionName="getCustomerProfile"
schemaURI="http://temp.openuri.org/DataServices/schemas/CustomerProfile
.xsd" schemaRootElement="CustomerProfile"
     */

org.openuri.temp.dataServices.schemas.customerProfile.CustomerProfileDo
cument getCustomerProfile(java.lang.String CustomerID);
```

6. Paste the copy on a new line and rename it getCustomerProfileWithFilter in the function definition.

```
    /**
     *
     * @jc:XDS
functionURI="ld:DataServices/CustomerManagement/CustomerProfile"
functionName="getCustomerProfile"
schemaURI="http://temp.openuri.org/DataServices/schemas/CustomerProfile
.xsd" schemaRootElement="CustomerProfile"
     */
org.openuri.temp.dataServices.schemas.customerProfile.CustomerProfileDo
cument getCustomerProfileWithFilter(java.lang.String CustomerID,
FilterXQuery filter);
```

7. Add the following parameter to the `getCustomerProfileWithFilter()` function:

```
FilterXQuery filter
```

After adding this parameter, the function signature will display as:

```
    /**
     *
     * @jc:XDS
functionURI="ld:DataServices/CustomerManagement/CustomerProfile"
functionName="getCustomerProfile"
schemaURI="http://temp.openuri.org/DataServices/schemas/CustomerProfile
.xsd" schemaRootElement="CustomerProfile"
```

```
     */
org.openuri.temp.dataServices.schemas.customerProfile.CustomerProfileDo
cument getCustomerProfileWithFilter(java.lang.String CustomerID,
FilterXQuery filter);
```

8.3 Inserting a Data Service Control into a Page Flow

At this point, you have created a Data Service Control and specified which data service functions (`getCustomerProfile()` and `submitCustomerProfile()`) you want to want to use in this control. However, the control is not yet associated with a page flow, from which end-users can retrieve data.

Objectives

In this exercise, you will:

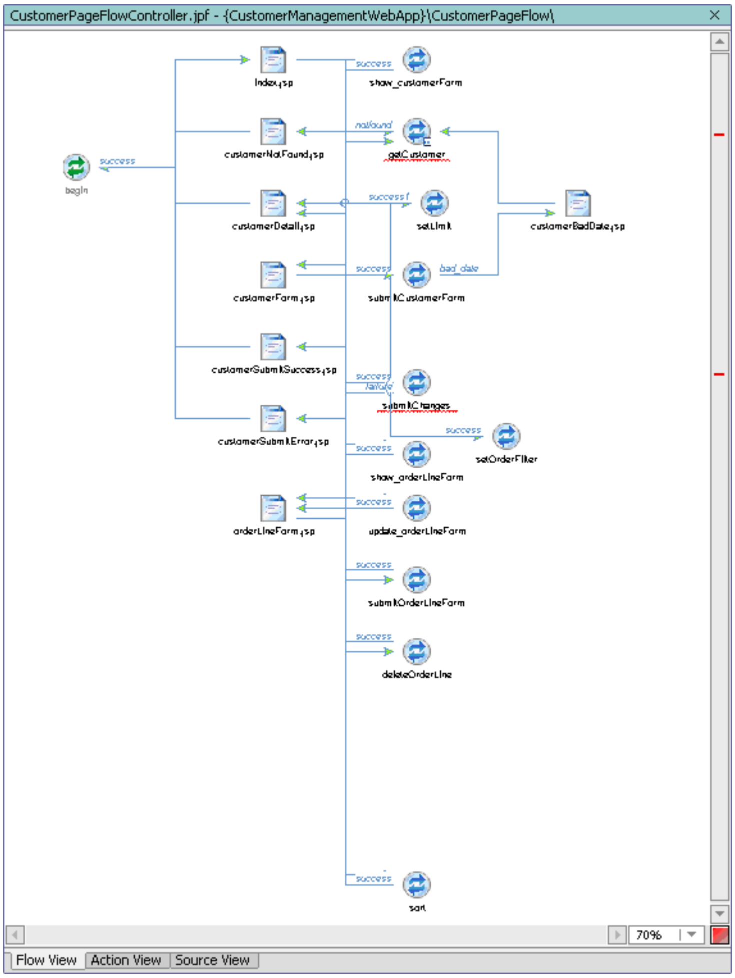
- Use Flow View to add the CustomerData control to the `CustomerPageFlowController.jpf` file.
- Use Source View to confirm the addition.

Instructions

1. Open `CustomerPageFlowController.jpf` in Flow View. (The file is located in the CustomerManagementWebApp\CustomerPageFlow folder.)

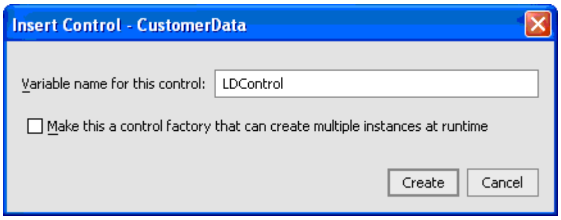
Note: There are two "errors" in the file, indicated by the two red marks in the scrollbar. This is because the `getCustomer()` and `submitCustomer()` functions are not yet associated with a Data Services Control.

Figure 8-5 Page Flow View



2. In Data Palette, go to Controls. (If Data Palette is not open, choose View → Windows → Data Palette.)
3. Choose Add → Local Controls → CustomerData, and name it LDControl.
4. Click Create.

Figure 8-6 Insert Custom Data Service Control


Insert Control - CustomerData
Variable name for this control:
LDControl
Make this a control factory that can create multiple instances at runtime
Create
Cancel

5. Open the `CustomerPageFlowController.jpf` in Source View.

6. Change the line:

```
customerDocument =
LDControl.gCustomerProfileWithFilter(form.getCustomerID(),filter);
```

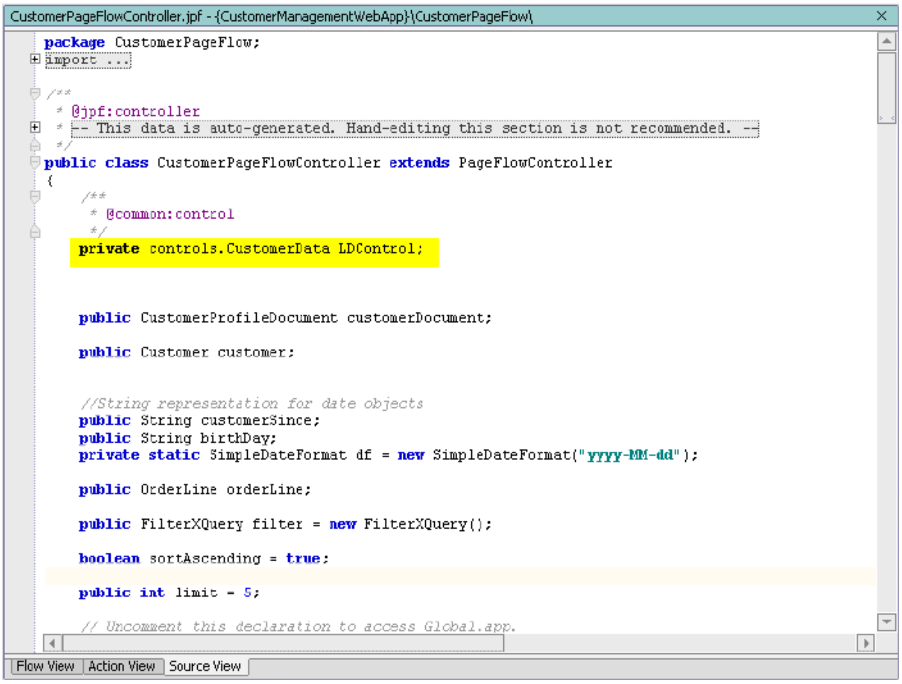
to:

```
customerDocument = LDControl.getCustomerProfile(form.getcustomerID());
```

7. Confirm that the page flow now includes the control as an instance variable:

```
private controls.CustomerData LDControl;
```

Figure 8-7 Source View of a Data Service Control


CustomerPageFlowController.jpf - {CustomerManagementWebApp}\CustomerPageFlow\
Flow View
Action View
Source View

```
package CustomerPageFlow;
import ...

/**
 * @jpf:controller
 * -- This data is auto-generated. Hand-editing this section is not recommended. --
 */
public class CustomerPageFlowController extends PageFlowController
{
    /**
     * @common:control
     */
    private controls.CustomerData LDControl;


    public CustomerProfileDocument customerDocument;

    public Customer customer;


    //String representation for date objects
    public String customerSince;
    public String birthDay;
    private static SimpleDateFormat df = new SimpleDateFormat("yyyy-MM-dd");

    public OrderLine orderLine;

    public FilterXQuery filter = new FilterXQuery();

    boolean sortAscending = true;

    public int limit = 5;

    // Uncomment this declaration to access Global.app.
```

8.4 Running the Web Application

In this exercise you will see the Data Service Control in action.

Objectives

In this exercise, you will:

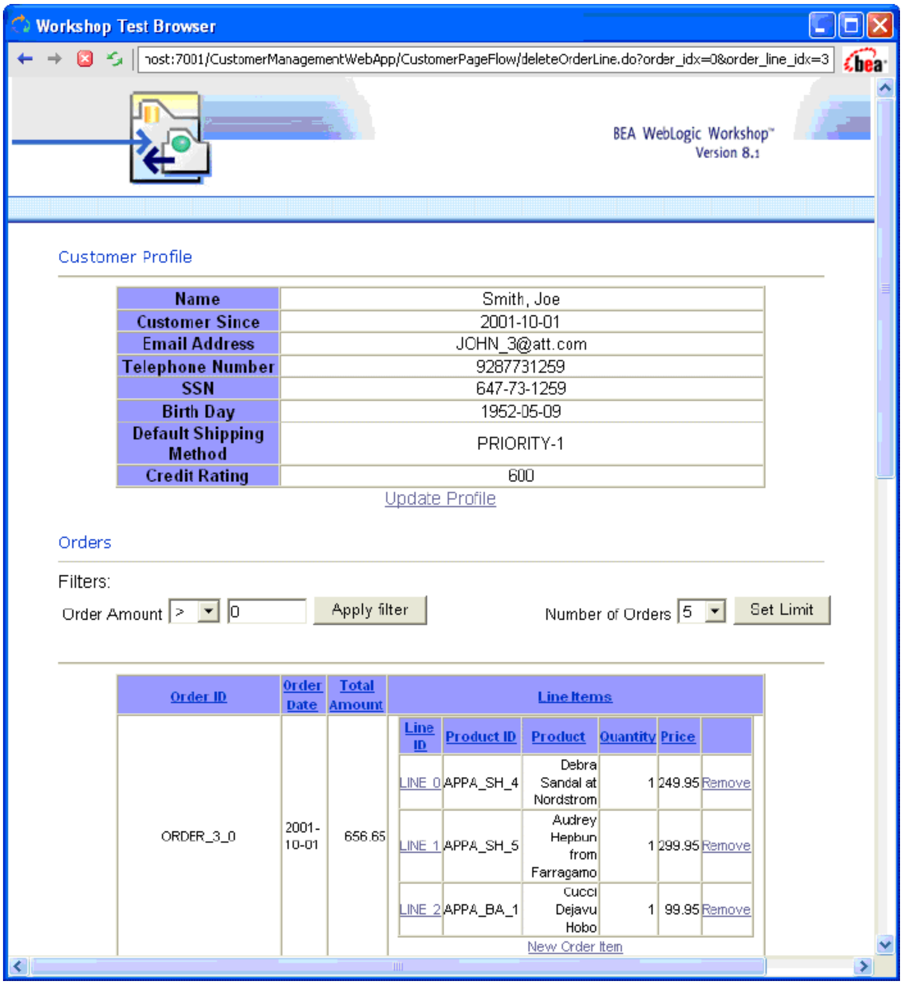
- Run the Web application, which now contains a Data Service Control.
- Use `getCustomerProfile()` to retrieve data about a specific customer.
- Use `submitCustomerProfile()` to update customer data.
- Use ALDSP Test View to confirm that changes were persisted.

Instructions

Note: The WebLogic Server must be running.

1. Build the CustomerManagementWebApp project.
2. Open `CustomerPageFlowController.jpf` in Flow View.
3. Click the Start icon (or press Ctrl + F5) to run the web application. The Workshop Test Browser opens after a few moments.
4. Enter CUSTOMER3 in the customer ID field and click Submit. The profile and order information for Britt Pierce should be returned.

Figure 8-8 Java Page Flow Results


Workshop Test Browser
host:7001/CustomerManagementWebApp/CustomerPageFlow/deleteOrderLine.do?order_idx=0&order_line_idx=3
BEA WebLogic Workshop
Version 8.1

Customer Profile

Name	Smith, Joe
Customer Since	2001-10-01
Email Address	JOHN_3@att.com
Telephone Number	9287731259
SSN	647-73-1259
Birth Day	1952-05-09
Default Shipping Method	PRIORITY-1
Credit Rating	600

Update Profile

Orders

Filters:

Order Amount > 0 Apply filter Number of Orders 5 Set Limit

Order ID	Order Date	Total Amount	Line ID	Product ID	Product	Quantity	Price	
ORDER_3_0	2001-10-01	656.65	LINE_0	APPA_SH_4	Debra Sandal at Nordstrom	1	249.95	Remove
			LINE_1	APPA_SH_5	Audrey Hepbun from Farragamo	1	299.95	Remove
			LINE_2	APPA_BA_1	Gucci Dejavu Hobo	1	99.95	Remove

New Order Item

Modify the customer information by completing the following steps:

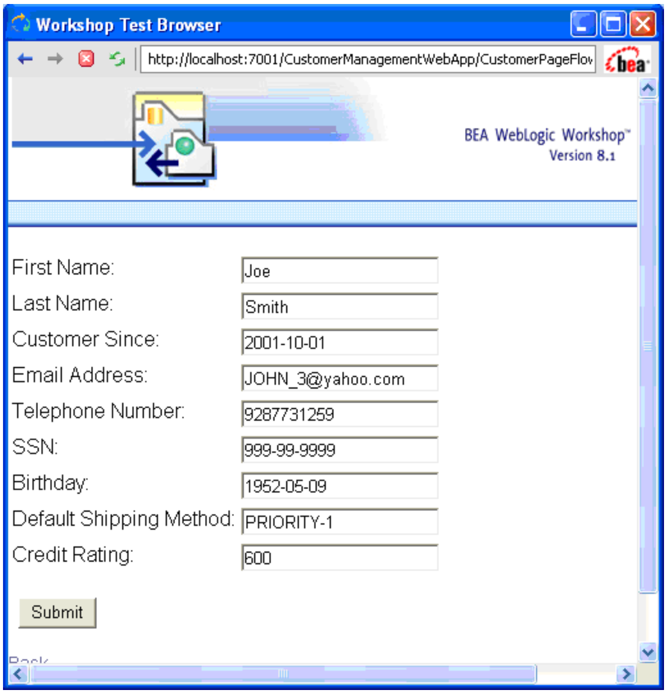
5. Click Update Profile.

    a. Modify Email Address to the following:

```
JOHN_3@yahoo.com
```

    b. Click Submit.

Figure 8-9 Updating a Customer Profile


Workshop Test Browser
http://localhost:7001/CustomerManagementWebApp/CustomerPageFlow
BEA WebLogic Workshop™
Version 8.1
Submit

First Name:	Joe
Last Name:	Smith
Customer Since:	2001-10-01
Email Address:	JOHN_3@yahoo.com
Telephone Number:	9287731259
SSN:	999-99-9999
Birthday:	1952-05-09
Default Shipping Method:	PRIORITY-1
Credit Rating:	600

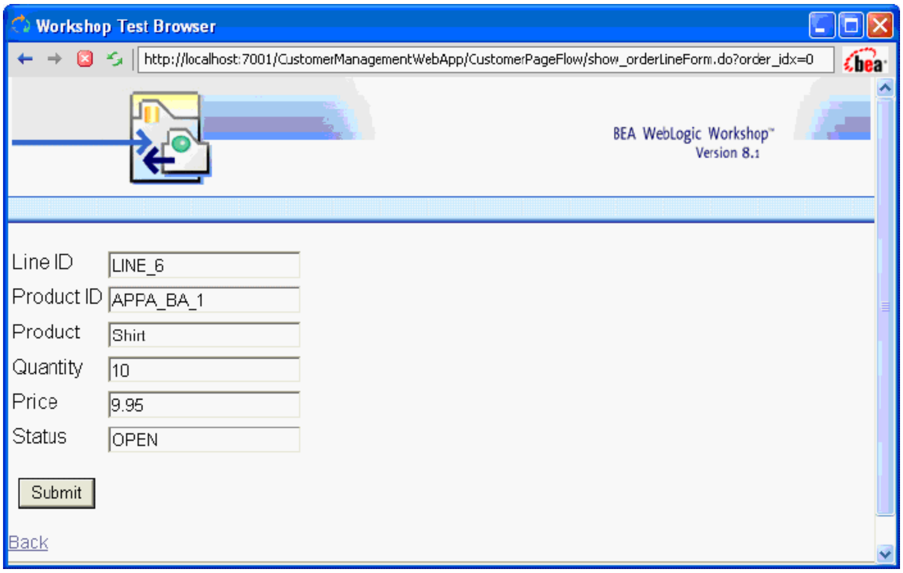
c. Click Submit All Changes. (The link is at the bottom of the Workshop Test Browser page.)

6. Add a new order line item by completing the following steps:

 a. In Order_3_0, click New Order Item. (The link is located at the bottom of all line items for Order_3_0.)

 b. Enter the new order information, as displayed in Figure 8-10, and then click Submit.

Figure 8-10 Adding New Order Information

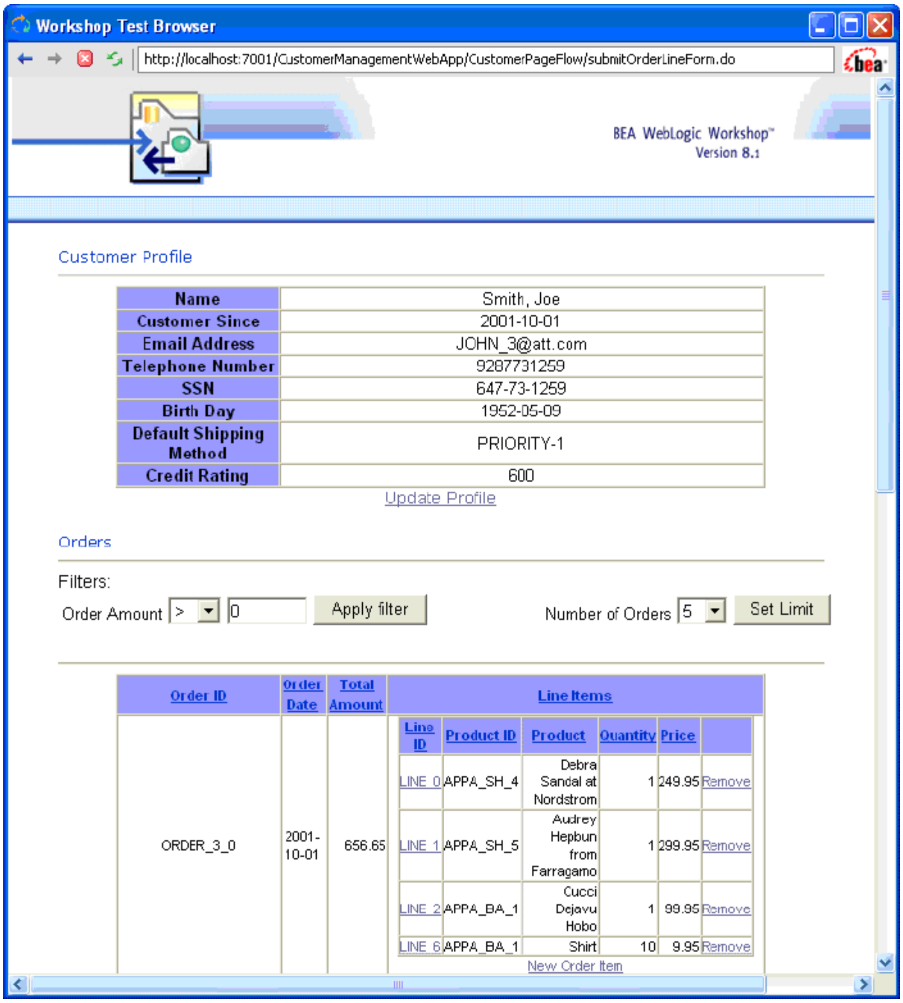


http://localhost:7001/CustomerManagementWebApp/CustomerPageFlow/show_orderLineForm.do?order_idx=0

Line ID	LINE_6
Product ID	APPA_BA_1
Product	Shirt
Quantity	10
Price	9.95
Status	OPEN

The new order information displays in the Workshop Test Browser.

Figure 8-11 Updated Data



Workshop Test Browser
http://localhost:7001/CustomerManagementWebApp/CustomerPageFlow/submitOrderLineForm.do
BEA WebLogic Workshop™ Version 8.1

Customer Profile

Name	Smith, Joe
Customer Since	2001-10-01
Email Address	JOHN_3@att.com
Telephone Number	9287731259
SSN	647-73-1259
Birth Day	1952-05-09
Default Shipping Method	PRIORITY-1
Credit Rating	600

Update Profile

Orders

Filters:

Order Amount > 0 Apply filter

Number of Orders 5 Set Limit

Order ID	Order Date	Total Amount	Line Items					
			Line ID	Product ID	Product	Quantity	Price	
ORDER_3_0	2001-10-01	656.65	LINE_0	APPA_SH_4	Debra Sandal at Nordstrom	1	249.95	Remove
			LINE_1	APPA_SH_5	Audrey Hepbun from Farragamo	1	299.95	Remove
			LINE_2	APPA_BA_1	Cucci Dejavu Hobo	1	99.95	Remove
			LINE_6	APPA_BA_1	Shirt	10	9.95	Remove
			New Order Item					

7. Modify an existing order by completing the following steps:

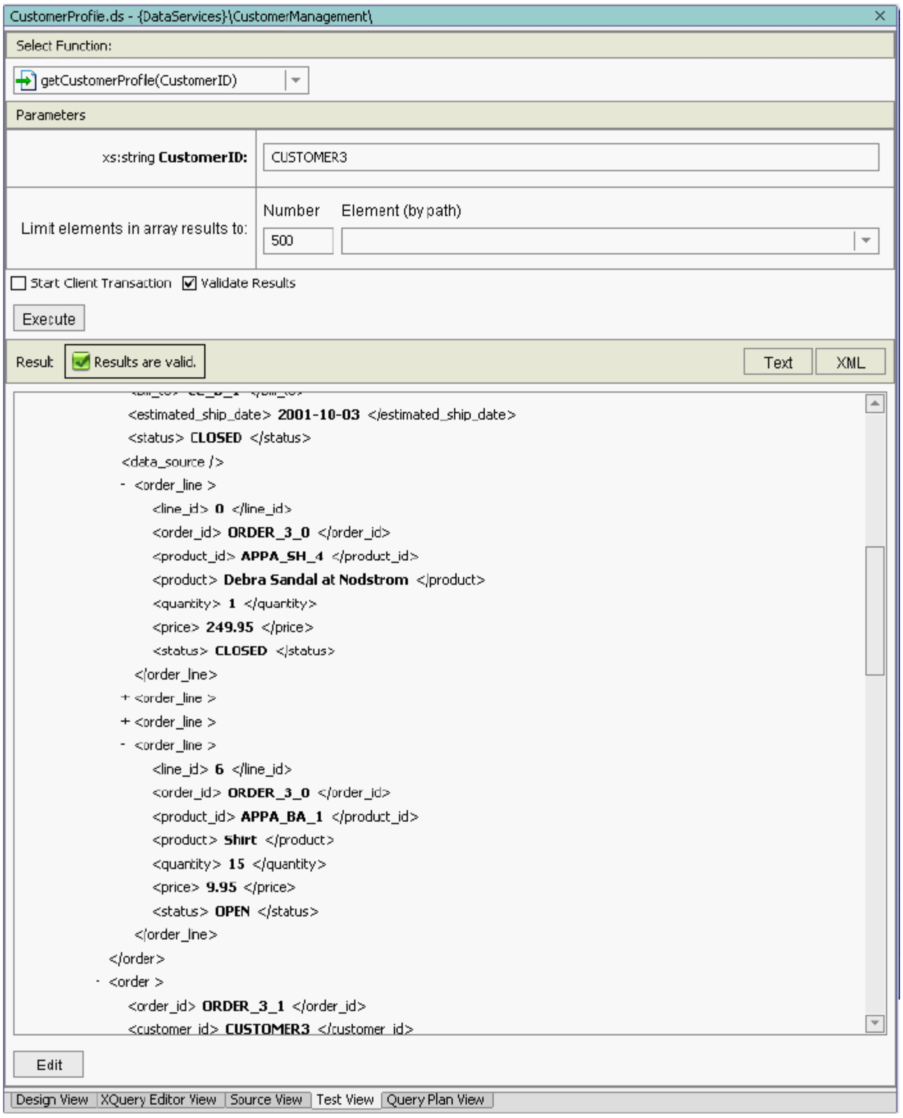
   a. In Order_3_0, click Line 6.

   b. Enter 15 in the Quantity field.

   c. Click Submit to close the Order Information window.

8. Click Submit All Changes. (The link is at the bottom of the Workshop Test Browser page.)
9. Close Workshop Test Browser.
10. Test whether the changes were persisted by completing the following steps:
    a. In WebLogic Workshop, open `CustomerProfile.ds` in Test View.
    b. Select `getCustomerProfile(CustomerID)` from the Function drop-down list.
    c. Enter CUSTOMER3 in the Parameter field.
    d. Click Execute.
    e. Expand the <creditrating>, <order> and <order_line> nodes to confirm that the changes persisted.

Figure 8-12 Test View -- Confirm Changes



CustomerProfile.ds - {DataServices}\CustomerManagement\
Select Function:
getCustomerProfile(CustomerID)
Parameters

xs:string **CustomerID:** CUSTOMER3

Limit elements in array results to: Number 500 Element (by path)

```
        <estimated_ship_date> 2001-10-03 </estimated_ship_date>
        <status> CLOSED </status>
       <data_source />
       - <order_line >
           <line_id> 0 </line_id>
           <order_id> ORDER_3_0 </order_id>
           <product_id> APPA_SH_4 </product_id>
           <product> Debra Sandal at Nodstrom </product>
           <quantity> 1 </quantity>
           <price> 249.95 </price>
           <status> CLOSED </status>
         </order_line>
       + <order_line >
       + <order_line >
       - <order_line >
           <line_id> 6 </line_id>
           <order_id> ORDER_3_0 </order_id>
           <product_id> APPA_BA_1 </product_id>
           <product> Shirt </product>
           <quantity> 15 </quantity>
           <price> 9.95 </price>
           <status> OPEN </status>
         </order_line>
     </order>
   - <order >
        <order_id> ORDER_3_1 </order_id>
        <customer_id> CUSTOMER3 </customer_id>
```

Lesson Summary

In this lesson, you learned how to:

- Install the Data Service Control in your application.
- Create a Data Service Control for a web project, and then add functions from your data service into the Data Service Control.
- Add the Data Service Control into a Java Page Flow.
- Use the Data Service Control to access data services from a web application.
- (Optional) Pass data service results to the JSP, using NetUI

Accessing Data Services Through Web Services

A Data Service Control can be used to access data through a page flow, Web service, or business logic. In the previous lesson, you created a Data Service Control and used it within a Web application's page flow. In this lesson, you will use that same Data Service Control to generate a `.wsdl` for a Web service that can invoke data service functions.

Objectives

After completing this lesson, you will be able to:

- Use a Data Service Control to generate a Web service for a data service.
- Test the generated Web service and invoke data service functions through the Web service interface.
- Generate a `.wsdl` file for Web service clients.

Overview

A Web service is a set of functions packaged into a single entity that is available to other systems on a network. The network can be a corporate intranet or the Internet. Other systems can call these functions to request data or perform an operation.

Web services are a useful way to provide data to an array of consumers over the Internet, like stock quotes and weather reports. But they take on a new power in the enterprise, where they offer a flexible solution for integrating distributed systems, whether legacy systems or new technology.

9.1 Generating a Web Service from a Data Service Control

In the previous lesson, you created a Data Service Control, which enabled WebLogic Workshop to generate a Java Control Extension (`.jcx`) file. This file contains the underlying data service's method calls. In this exercise, you will use that Data Service Control to generate a Web service.

Objectives

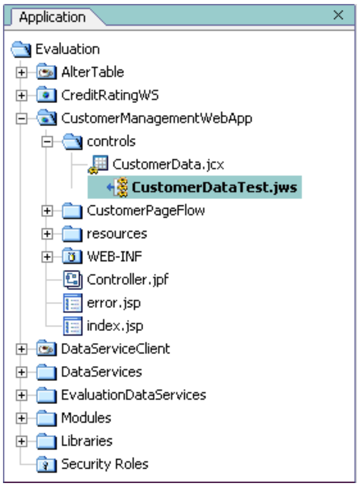
In this exercise, you will:

- Generate a stateless Web service interface, through which you can access the Data Service Control.
- Test the Web service to determine that it returns customer profile and order information.

Instructions

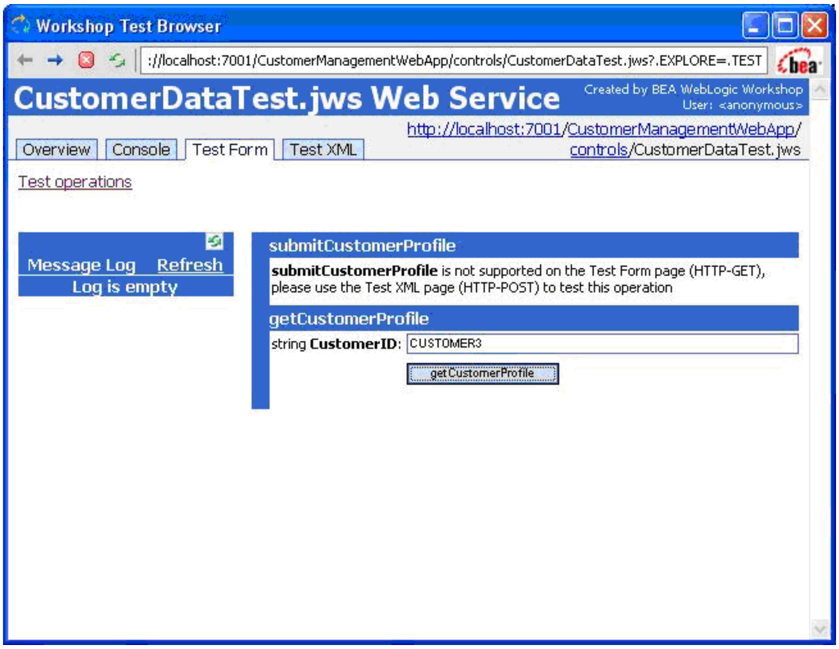
1. Expand the CustomerManagementWebApp and controls folders.
2. Right-click the CustomerData.jcx control.
3. Choose Generate Test JWS (Stateless). A new file, CustomerDataTest.jws, is generated. With this Java Web Service (`.jws`) file, the Data Service Control methods are now available through a Web service interface.

Figure 9-1 Java Web Service File



4. Open the `CustomerDataTest.jws` file in Source View.
5. Click the Start icon (or press Ctrl+F5). Workshop Test Browser opens.
6. Enter CUSTOMER3 in the string CUSTOMER ID field.

Figure 9-2 Workshop Test Browser: Web Service

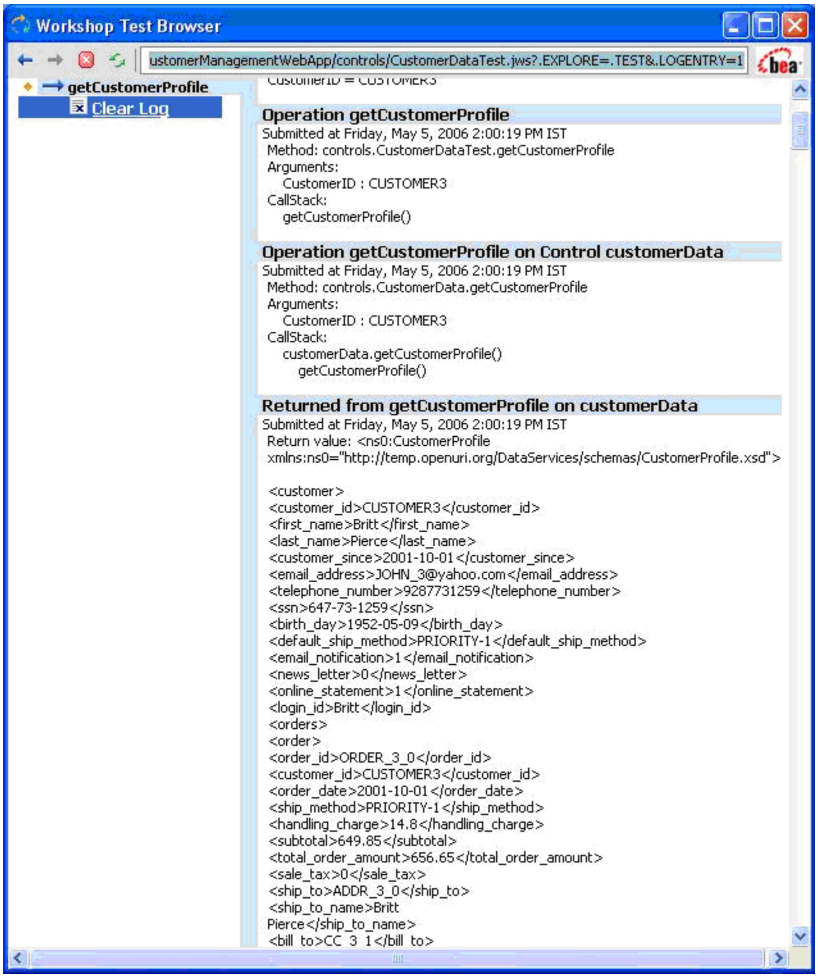

Workshop Test Browser
://localhost:7001/CustomerManagementWebApp/controls/CustomerDataTest.jws?.EXPLORE=.TEST
CustomerDataTest.jws Web Service
Created by BEA WebLogic Workshop
User: <anonymous>
http://localhost:7001/CustomerManagementWebApp/controls/CustomerDataTest.jws
Overview
Console
Test Form
Test XML
Test operations
Message Log
Refresh
Log is empty
submitCustomerProfile
getCustomerProfile
getCustomerProfile

submitCustomerProfile is not supported on the Test Form page (HTTP-GET), please use the Test XML page (HTTP-POST) to test this operation

string **CustomerID**: CUSTOMER3

7. Click getCustomerProfile. The customer profile and order information for Customer 3 is retrieved.
8. View both the "Returned from" and "Service Response" results, which should be similar to that displayed in Figure 9-3.

Figure 9-3 Web Service Test Results


Workshop Test Browser
ustomerManagementWebApp/controls/CustomerDataTest.jws?.EXPLORE=.TEST&.LOGENTRY=1
getCustomerProfile
Clear Log

```
CustomerID = CUSTOMER3

Operation getCustomerProfile
Submitted at Friday, May 5, 2006 2:00:19 PM IST
 Method: controls.CustomerDataTest.getCustomerProfile
 Arguments:
    CustomerID : CUSTOMER3
 CallStack:
    getCustomerProfile()

Operation getCustomerProfile on Control customerData
Submitted at Friday, May 5, 2006 2:00:19 PM IST
 Method: controls.CustomerData.getCustomerProfile
 Arguments:
    CustomerID : CUSTOMER3
 CallStack:
    customerData.getCustomerProfile()
       getCustomerProfile()

Returned from getCustomerProfile on customerData
Submitted at Friday, May 5, 2006 2:00:19 PM IST
 Return value: <ns0:CustomerProfile
 xmlns:ns0="http://temp.openuri.org/DataServices/schemas/CustomerProfile.xsd">

 <customer>
 <customer_id>CUSTOMER3</customer_id>
 <first_name>Britt</first_name>
 <last_name>Pierce</last_name>
 <customer_since>2001-10-01</customer_since>
 <email_address>JOHN_3@yahoo.com</email_address>
 <telephone_number>9287731259</telephone_number>
 <ssn>647-73-1259</ssn>
 <birth_day>1952-05-09</birth_day>
 <default_ship_method>PRIORITY-1</default_ship_method>
 <email_notification>1</email_notification>
 <news_letter>0</news_letter>
 <online_statement>1</online_statement>
 <login_id>Britt</login_id>
 <orders>
 <order>
 <order_id>ORDER_3_0</order_id>
 <customer_id>CUSTOMER3</customer_id>
 <order_date>2001-10-01</order_date>
 <ship_method>PRIORITY-1</ship_method>
 <handling_charge>14.8</handling_charge>
 <subtotal>649.85</subtotal>
 <total_order_amount>656.65</total_order_amount>
 <sale_tax>0</sale_tax>
 <ship_to>ADDR_3_0</ship_to>
 <ship_to_name>Britt
 Pierce</ship_to_name>
 <bill_to>CC_3_1</bill_to>
```

9. Close Workshop Test Browser.

9.2 Using a Data Service Control to Generate a WSDL for a Web Service

You can use the Java Web Service file to generate a WSDL. A WSDL file contains all of the information necessary for a client to invoke the methods of a Web service:

- The data types used as method parameters or return values.
- The individual methods names and signatures (WSDL refers to methods as operations).
- The protocols and message formats allowed for each method.
- The URLs used to access the Web service.

Objectives

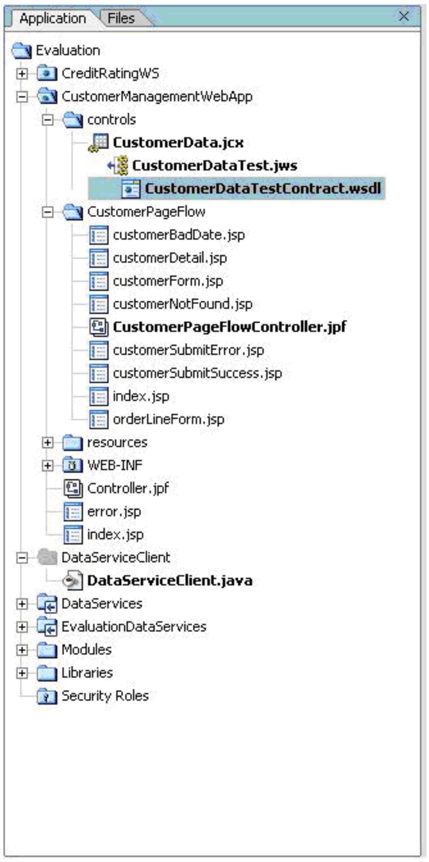
In this exercise, you will:

- Generate a `.wsdl` file, based on the Data Service Control.
- (Optional) View the `.wsdl` file's structure and source code.

Instructions

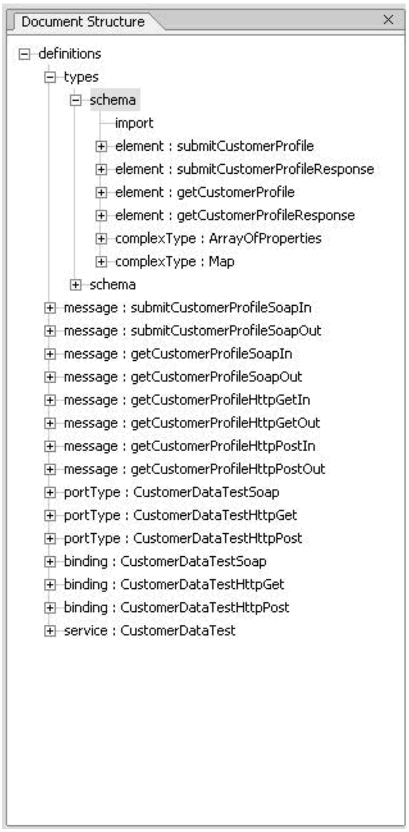
1. Right-click the CustomerDataTest.jws control.
2. Choose Generate WSDL File. The CustomerDataTestContract.wsdl is generated, which can be used by other Web service clients.

Figure 9-4 New WSDL File



3. (*Optional*) Open the `CustomerDataTestContract.wsdl` file and explore the document structure and source code.

Figure 9-5 Document Structure



```
⊟ definitions
    ⊟ types
        ⊟ schema
            import
            ⊞ element : submitCustomerProfile
            ⊞ element : submitCustomerProfileResponse
            ⊞ element : getCustomerProfile
            ⊞ element : getCustomerProfileResponse
            ⊞ complexType : ArrayOfProperties
            ⊞ complexType : Map
        ⊞ schema
    ⊞ message : submitCustomerProfileSoapIn
    ⊞ message : submitCustomerProfileSoapOut
    ⊞ message : getCustomerProfileSoapIn
    ⊞ message : getCustomerProfileSoapOut
    ⊞ message : getCustomerProfileHttpGetIn
    ⊞ message : getCustomerProfileHttpGetOut
    ⊞ message : getCustomerProfileHttpPostIn
    ⊞ message : getCustomerProfileHttpPostOut
    ⊞ portType : CustomerDataTestSoap
    ⊞ portType : CustomerDataTestHttpGet
    ⊞ portType : CustomerDataTestHttpPost
    ⊞ binding : CustomerDataTestSoap
    ⊞ binding : CustomerDataTestHttpGet
    ⊞ binding : CustomerDataTestHttpPost
    ⊞ service : CustomerDataTest
```

Lesson Summary

In this lesson, you learned how to:

- Use a Data Service Control to generate a Web service for a data service.
- Test the generated Web service and invoke data service functions through the Web service interface.
- Generate a `.wsdl` file for Web service clients.

Updating Data Services Using Java

One of the features introduced with Data Services Platform (ALDSP) is the ability to write data back to the underlying data sources. This write service is built on top of the Service Data Object (SDO) specification, and provides the ability to update, insert, and delete results returned by a data service. It also provides the ability to submit all changes to the SDO (inserts, deletes, and updates) to the underlying data sources for persisting.

Objectives

After completing this lesson, you will be able to:

- Update, add to, and delete data from data service objects.
- Submit changes to the underlying data sources, using the Mediator API.

Overview

When you update, add, or delete from data service objects, all changes are logged in the SDO's change summary.

10.1 Modifying and Saving Changes to the Underlying Data Source

Although the steps in the next three exercises are different, the underlying principle is the same: When you update, add, or delete from data service objects, all changes are logged in the SDO's change summary. When the change is submitted, items indicated in the Change Summary log are applied in a transactionally-safe manner, and then persisted to the underlying data source. Changes to relational data sources are automatically applied, while changes to other data services, such as Web services and portals, are applied using a ALDSP update framework.

Objectives

In this exercise, you will:

- Modify customer data and save the changes to the SDO Change Summary log.
- View the results in the Output window.
- Invoke the `submit()` method of the Mediator API to save the changes to the underlying data source.
- Verify the results in Test View.

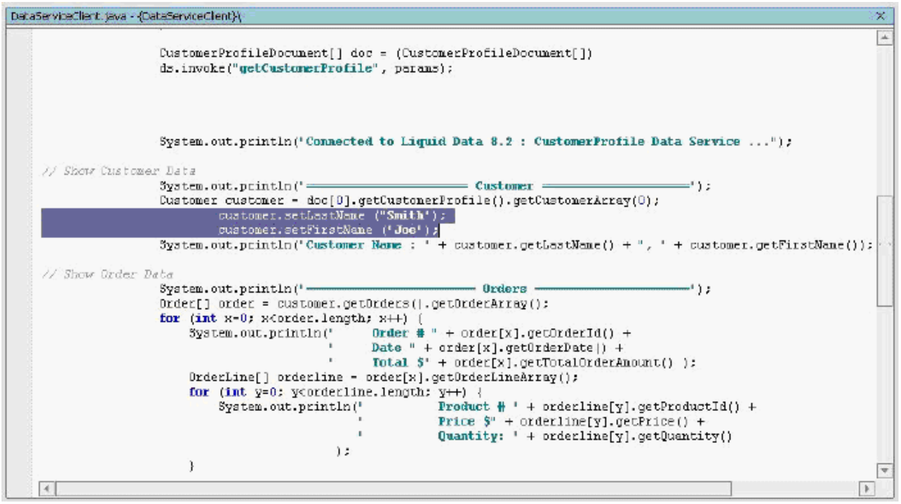
Instructions

1. Open the `DataServiceClient.java` file, located in the DataServiceClient project folder.
2. Change the first and last name of CUSTOMER3 from Brett Pierce to Joe Smith, by using the `set()` methods of the Customer data object instance. You do this by adding the `set()` method to the //Show Customer Data section (new code is displayed in boldface type):

```
Customer customer = doc[0].getCustomerProfile().getCustomerArray(0);
customer.setLastName("Smith");
customer.setFirstName("Joe");
System.out.println("Customer Name: " + customer.getLastName() +
", " + customer.getFirstName());
```

Note: The Array of function has been deprecated. Ensure that you modify `doc.getCustomerProfile().getCustomerArray(0)` to `doc[0].getCustomerProfile().getCustomerArray(0):`

Figure 10-1 Set() Method Specified



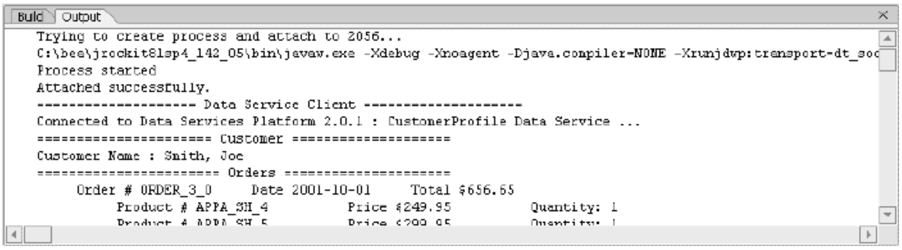
DataServiceClient.java - {DataServiceClient}\

```
                    CustomerProfileDocument[] doc = (CustomerProfileDocument[])
                    ds.invoke("getCustomerProfile", params);



                    System.out.println("Connected to Liquid Data 8.2 : CustomerProfile Data Service ...");

// Show Customer Data
                    System.out.println("================== Customer ==================");
                    Customer customer = doc[0].getCustomerProfile().getCustomerArray(0);
                            customer.setLastName ("Smith");
                            customer.setFirstName ("Joe");
                    System.out.println("Customer Name : " + customer.getLastName() + ", " + customer.getFirstName());

// Show Order Data
                    System.out.println("================== Orders ==================");
                    Order[] order = customer.getOrders().getOrderArray();
                    for (int x=0; x<order.length; x++) {
                        System.out.println("      Order # " + order[x].getOrderId() +
                                         "      Date " + order[x].getOrderDate() +
                                         "      Total $" + order[x].getTotalOrderAmount() );
                        OrderLine[] orderline = order[x].getOrderLineArray();
                        for (int y=0; y<orderline.length; y++) {
                            System.out.println("          Product # " + orderline[y].getProductId() +
                                             "          Price $" + orderline[y].getPrice() +
                                             "          Quantity: " + orderline[y].getQuantity()
                                         );
                        }
```

3. Save your work.
4. Right-click the DataServiceClient project folder and choose Build DataServiceClient.
5. Click the `DataServiceClient.java` file's Start icon (or press Ctrl + F5).
6. Confirm that the changes were submitted, by viewing the results in the Output window. (If the window is not open, choose View → Windows → Output.)

   Note: At this point, the changes only exist as entries in the SDO Change Summary Log, not in the data source. You must complete the remaining steps in this exercise to ensure that the underlying data source is updated.

Figure 10-2 Change Results in Output Window



```
Trying to create process and attach to 2056...
C:\bea\jrockit81sp4_142_05\bin\javaw.exe -Xdebug -Xnoagent -Djava.compiler=NONE -Xrunjdwp:transport=dt_soc
Process started
Attached successfully.
-------------------- Data Service Client --------------------
Connected to Data Services Platform 2.0.1 : CustomerProfile Data Service ...
====================== Customer ====================
Customer Name : Smith, Joe
======================= Orders ====================
     Order # ORDER_3_0     Date 2001-10-01     Total $656.65
          Product # APPA_SH_4          Price $249.95          Quantity: 1
          Product # APPA_SH_5          Price $299.95          Quantity: 1
```

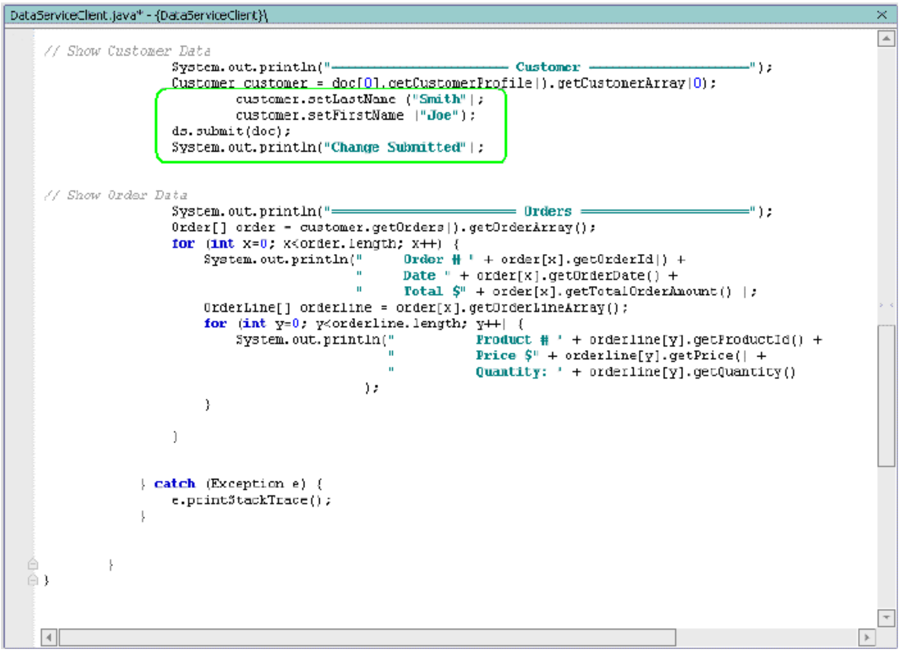
7. Invoke the Mediator API's `submit()` method and save the changes to the data source, by using the data service instance. The `submit()` method takes two parameters: the document to submit and the data service name. You do this by adding the following code into the //Show Customer Data section of the file:

   ```
   ds.submit(doc);
   ```

8. Change the output code, as follows:

   ```
   System.out.println("Change Submitted");
   ```

Figure 10-3 submit() and Output Method Specified



DataServiceClient.java* - {DataServiceClient}\

```
// Show Customer Data
                    System.out.println("---------------------- Customer ---------------------");
                    Customer customer = doc[0].getCustomerProfile().getCustomerArray(0);
                              customer.setLastName ("Smith");
                              customer.setFirstName ("Joe");
                    ds.submit(doc);
                    System.out.println("Change Submitted");

// Show Order Data
                    System.out.println("====================== Orders =====================");
                    Order[] order = customer.getOrders().getOrderArray();
                    for (int x=0; x<order.length; x++) {
                         System.out.println("     Order # " + order[x].getOrderId() +
                                            "     Date " + order[x].getOrderDate() +
                                            "     Total $" + order[x].getTotalOrderAmount() );
                         OrderLine[] orderline = order[x].getOrderLineArray();
                         for (int y=0; y<orderline.length; y++) {
                              System.out.println("          Product # " + orderline[y].getProductId() +
                                                 "          Price $" + orderline[y].getPrice() +
                                                 "          Quantity: " + orderline[y].getQuantity()
                                                 );
                         }

                    }


               } catch (Exception e) {
                    e.printStackTrace();
               }


          }
}
```

9. Click the `DataServiceClient.java` file's Start icon (or press Ctrl + F5).

10. Open `DataServices\CustomerManagement\CustomerProfile.ds` in Test View.

11. Select the `getCustomerProfile(CustomerID)` function.

12. Enter CUSTOMER3 in the xs:string CustomerID field.

13. Click Execute. The results should show the customer name as Joe Smith.

10.2 Inserting New Data to the Underlying Data Source Using Java

You can use the Mediator API to add new information to the underlying data source, thereby reducing the need to know a variety of data source APIs.

Objectives

In this exercise, you will:

- Add new data and save the changes to the SDO Change Summary log.
- Invoke the `submit()` method of the Mediator API to save the changes to the underlying data source.
- Verify the results in Test View.

Instructions

1. In WebLogic Workshop open the `DataServiceClient.java` file.
2. Add a new item to ORDER_3_0 (the first order placed by CUSTOMER3), by using the `addNewOrderLine()` method of the Order Item data object instance. You do this by inserting the following code into the //Show Customer Data section, after `System.out.println("Change Submitted"):`

```
// Get the order

     Order myorder = customer.getOrders().getOrderArray(0);

     // Create a new order item

     OrderLine newitem = myorder.addNewOrderLine();
```

3. Set the values of the new order item, including values for all required columns. (You can check the physical or logical `.xsd` file to determine what elements are required.) All foreign keys must be valid; therefore, use APPA_GL_3 as the Product ID.

   You do not need to `setOrderID()`; the SDO update will automatically set the foreign key to match its parent because the item will be added as a child of ORDER_3_0.

   To set the values, insert the following code above the //Show Order Data section of the Java file:

```
// Fill the values of the new order item

     newitem.setLineId("8");
```

```
        newitem.setProductId("APPA_GL_3");
        newitem.setProduct("Shirt");
        newitem.setQuantity(new BigDecimal(10));
        newitem.setPrice(new BigDecimal(10));
        newitem.setStatus("OPEN");
```

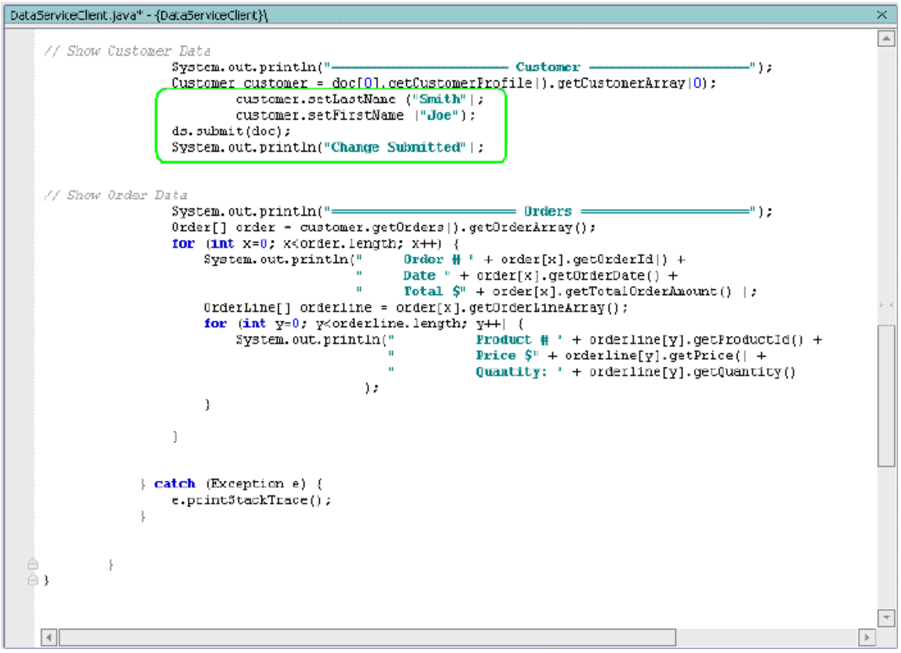
4. Press Alt + Enter to enable `java.math.BigDecimal`.
5. Invoke the Mediator API's submit method and save the changes to the data source, by using the data service instance. (The `submit()` method takes: the document to submit as a parameter)

   You do this by inserting the following code before the //Show Order Data section of the java file:

```
// Submit new order item
        ds.submit(doc,
"ld:DataServices/CustomerManagement/CustomerProfile.ds");
        System.out.println("Change Submitted");
```

6. Comment out the code where customer first name and last name were set, including call to submit method
7. Confirm that the //Show Customer Data section of your java file is as displayed in Figure 10-4.

Figure 10-4 xJava Code to Add Line Item


DataServiceClient.java* - {DataServiceClient}\

```
// Show Customer Data
                    System.out.println("———————————————— Customer ————————————————");
                    Customer customer = doc[0].getCustomerProfile().getCustomerArray(0);
                              customer.setLastName ("Smith");
                              customer.setFirstName ("Joe");
                    ds.submit(doc);
                    System.out.println("Change Submitted");

// Show Order Data
                    System.out.println("================ Orders ================");
                    Order[] order = customer.getOrders().getOrderArray();
                    for (int x=0; x<order.length; x++) {
                         System.out.println("     Order # ' + order[x].getOrderId() +
                                        "     Date " + order[x].getOrderDate() +
                                        "     Total $" + order[x].getTotalOrderAmount() );
                         OrderLine[] orderline = order[x].getOrderLineArray();
                         for (int y=0; y<orderline.length; y++) {
                              System.out.println("          Product # ' + orderline[y].getProductId() +
                                             "          Price $" + orderline[y].getPrice() +
                                             "          Quantity: ' + orderline[y].getQuantity()
                                        );
                         }

                    }


               } catch (Exception e) {
                    e.printStackTrace();
               }


          }
}
```

8. Open `DataServices\CustomerManagement\ CustomerProfile.ds` in TestView.

9. Enter CUSTOMER3 in the xs:string CustomerID field.

10. Click Execute. The result should contain the new order information.

10.3 Deleting Data from the Underlying Data Source Using Java

You can use the Mediator API to delete information to the underlying data source, thereby reducing the need to know a variety of data source APIs.

Objectives

In this exercise, you will:

- Delete data and save the changes to the SDO Change Summary log.
- Invoke the `submit()` method of the Mediator API to save the changes to the underlying data source.
- Verify the results in Test View.

Instructions

1. In Workshop Test Browser, determine the new item's placement in the array and subtract 1. For example, if line item with line_id = 8 is the fifth item for ORDER_3_0, its order placement is 4.
2. Close Workshop Test Browser.
3. In the `DataServicesClient.java` file delete or comment out the code that added a new order line item.
4. Add an instance of the item that you want to delete, by inserting the following code file:

```
// Get the order item

     OrderLine myItem =
customer.getOrders().getOrderArray(0).getOrderLineArray(4);
```

Note: The `getOrderLineArray()` is based on the item's placement in the array. In this case, 8 is the fifth item, making the variable 4. You should use the variable that is correct for your situation.

5. Call the delete method by inserting the following code:

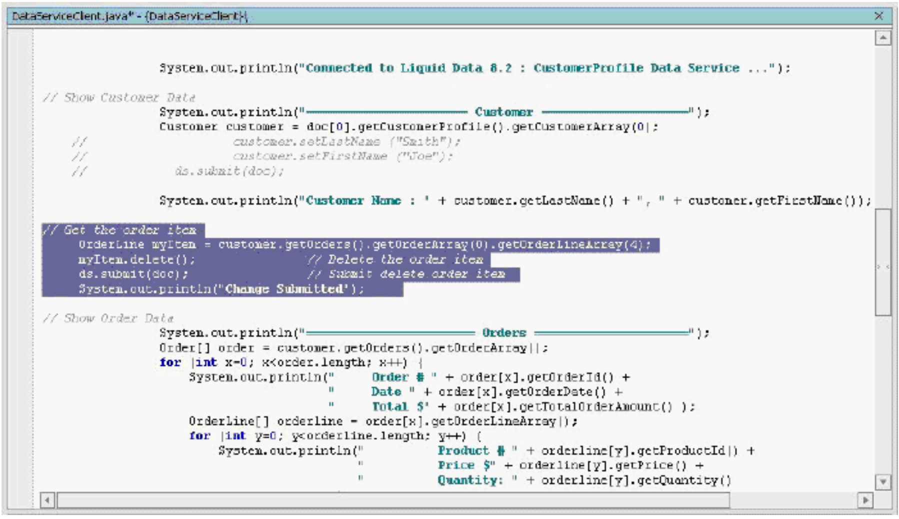
```
// Delete the order item

myItem.delete();
```

6. Submit the changes, using the Mediator API's `submit()` method.

```
// Submit delete order item

" ds.submit(doc);

    System.out.println("Change Submitted");
```

7. Confirm that the code is as displayed in Figure 10-5.

Figure 10-5 Java Code to Delete Line Item



DataServiceClient.java* - {DataServiceClient}

```
                    System.out.println("Connected to Liquid Data 8.2 : CustomerProfile Data Service ...");

// Show Customer Data
                    System.out.println("———————————————— Customer ———————————————");
                    Customer customer = doc[0].getCustomerProfile().getCustomerArray(0);
    //                          customer.setLastName ("Smith");
    //                          customer.setFirstName ("Joe");
    //                    ds.submit(doc);

                    System.out.println("Customer Name : " + customer.getLastName() + ", " + customer.getFirstName());

// Get the order item
     OrderLine myItem = customer.getOrders().getOrderArray(0).getOrderLineArray(4);
     myItem.delete();                // Delete the order item
     ds.submit(doc);                 // Submit delete order item
     System.out.println("Change Submitted");

// Show Order Data
                    System.out.println("———————————————— Orders ———————————————");
                    Order[] order = customer.getOrders().getOrderArray();
                    for (int x=0; x<order.length; x++) {
                        System.out.println("     Order # " + order[x].getOrderId() +
                                           "     Date " + order[x].getOrderDate() +
                                           "     Total $" + order[x].getTotalOrderAmount() );
                        OrderLine[] orderline = order[x].getOrderLineArray();
                        for (int y=0; y<orderline.length; y++) {
                            System.out.println("          Product # " + orderline[y].getProductId() +
                                               "          Price $" + orderline[y].getPrice() +
                                               "          Quantity: " + orderline[y].getQuantity()
```

8. Build the DataServiceClient project.

9. Click the `DataServiceClient.java` file's Start icon (or press Ctrl + F5) to run the program.

10. Confirm that the changes persisted to the underlying data source by completing the following steps:

    a. Click the `CustomerPageFlowController.jpf` application's Start icon (or press Ctrl+F5) to open the Workshop Test Browser.

    b. In the Workshop Test Browser, enter CUSTOMER3 in the Customer ID field and click Submit.

    c. Find ORDER_3_0 and verify that Line 8 is no longer present.

    d. Close the Workshop Test Browser

Lesson Summary

In this lesson, you learned how to:

- Update, add to, and delete data from data service objects.
- Submit changes to the underlying data sources, using the Mediator API.

Filtering, Sorting, and Truncating XML Data

When designing your data service, you can specify read functions that filter data service return values. However, instead of trying to create a read function for every possible client requirement, you can create generalized read functions to which client applications can apply custom filtering or ordering criteria at runtime.

Objectives

After completing this lesson, you will be able to:

- Use the FilterXQuery class to create dynamic filter, sort, and truncate data service results.
- Apply the FilterXQuery class to a data service, using the Mediator API or Data Service Control.

Overview

Data users often want to access information in ways that are not anticipated in the design of a data service. The filtering and ordering API allow client applications to control what data is returned by a data service read function call based on conditions specified at runtime.

Although you can specify read functions that filter data service return values, it may be difficult to anticipate all the ways that client applications may want to filter return values. To deal with this contingency, ALDSP lets client applications specify dynamic filtering, sorting, and truncating criteria against the data service. These criteria are evaluated on the Server, before being transmitted on the network, thereby reducing the data set results to items matching the criteria. Where possible, these instances are “pushed down” to the underlying data source, thereby reducing the data set returned to the user.

The advantage of the FilterXQuery class is that you can define client-side filtering operations, without modifying or re-deploying your data services.

11.1 Filtering Data Service Results

With the FilterXQuery class addFilter() method, filtering criteria are specified as Boolean condition statements (for example, ORDER_AMOUNT > 1000). Only items that meet the condition are included in the return set.

The `addFilter()` method also lets you create compound filters that provide significant flexibility, given the hierarchical structure of the data service return type. In other words, given a condition on a nested element, compound filters let you control the effects of the condition in relation to the parent element.

For example, consider a multi-level data hierarchy for CUSTOMERS/CUSTOMER/ORDER, in which CUSTOMERS is the top level document element, and CUSTOMER and ORDER are sequences within CUSTOMERS and CUSTOMER respectively. Finally, ORDER_AMOUNT is an element within ORDER.

An ORDER_AMOUNT condition (for example, CUSTOMER/ORDER/ORDER_AMOUNT > 1000) can affect what values are returned in several ways:

- It can cause all CUSTOMER objects to be returned, but filter ORDERS that have an amount less than 1000.
- It can cause only CUSTOMER objects to be returned that have at least one large order. All ORDER objects are returned for every CUSTOMER.
- It can cause only CUSTOMER objects to be returned that have at least one large order along with only large ORDER objects.
- It can cause only CUSTOMER objects to be returned for which every ORDER is greater than 1000.

Instead of writing XQuery functions for each case, you just pass the filter object as a parameter when executing a data service function, either using the Data Service Control or Mediator API.

Objectives

In this exercise, you will:

- Import the FilterXQuery class, which enables filtering, truncating, and sorting of data.

- Add a condition filter.
- View the results through the Mediator API.

Instructions

1. Open the DataServiceClient.java file.
2. Delete the code that removed the line item with line_id = 8 order item delete code.
3. Delete the invoke and println code from the //Insert Code section:

```
CustomerProfileDocument[] doc = (CustomerProfileDocument[])
ds.invoke("getCustomerProfile",params);

System.out.println("Connected to Liquid Data 8.2 : CustomerProfile Data
Service ...");
```

4. Import the FilterXQuery class by adding the following code:

```
import com.bea.ld.filter.FilterXQuery;
import com.bea.dsp.RequestConfig;
```

5. Create a filter instance of the FilterXQuery, plus specify a condition to filter orders greater than $1,000, by adding the following code:

```
//Create a filter and condition

FilterXQuery filter = new FilterXQuery();

filter.addFilter(

"CustomerProfile/customer/orders/order",
"CustomerProfile/customer/orders/order/total_order_amount",

">", "1000");
```

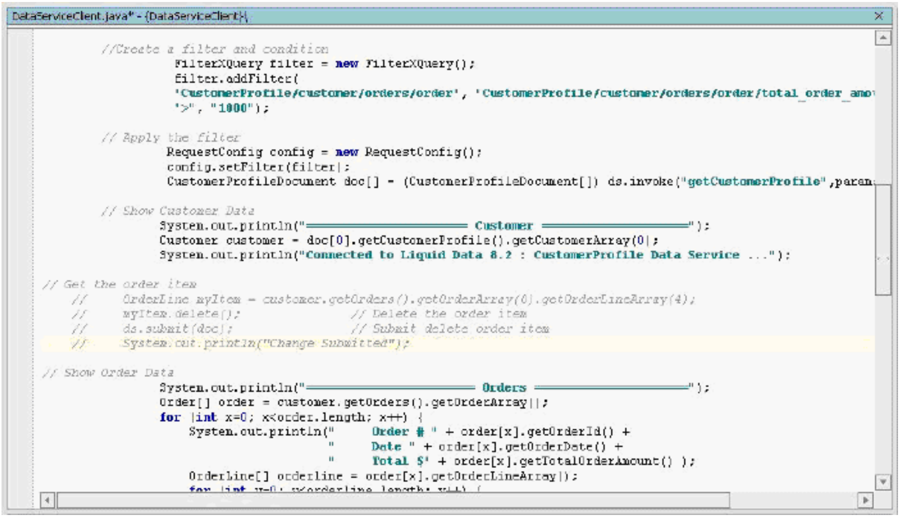
6. Apply the filter to the data service, by adding the following code:

```
// Apply the filter

  RequestConfig config = new RequestConfig();

  config.setFilter(filter);

  CustomerProfileDocument doc[] = (CustomerProfileDocument[])
ds.invoke("getCustomerProfile",params, config);
```

7. Change the `//Show Customer Data` code to the following:

```
// Show Customer Data

    System.out.println("======================= Customers
=====================");

    Customer customer = doc[0].getCustomerProfile().getCustomerArray(0);

    System.out.println("Connected to ALDSP: CustomerProfile Data Service
...");
```

Figure 11-1 Filter Code



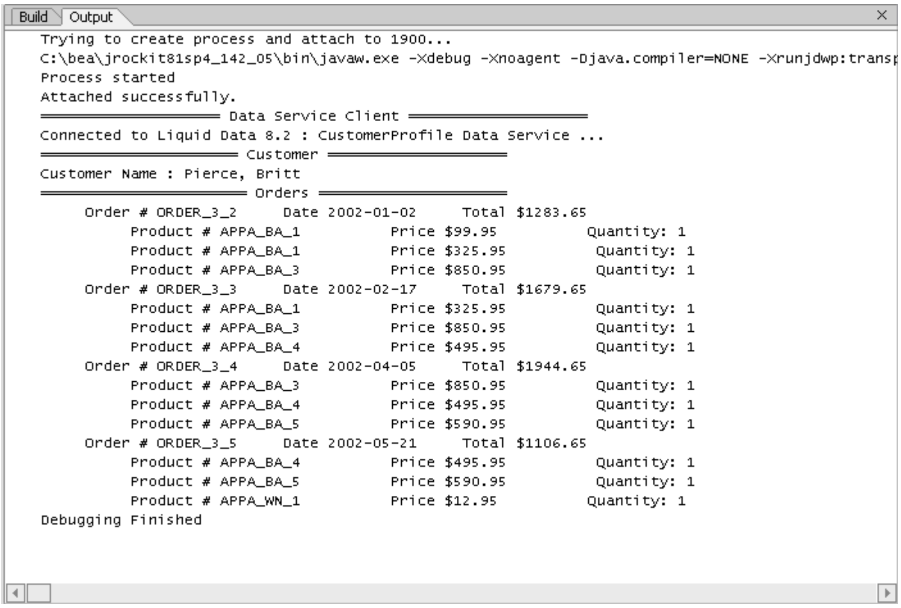
DataServiceClient.java* - {DataServiceClient}

```
        //Create a filter and condition
                FilterXQuery filter = new FilterXQuery();
                filter.addFilter(
                'CustomerProfile/customer/orders/order', 'CustomerProfile/customer/orders/order/total_order_amo
                '>", "1000");

        // Apply the filter
               RequestConfig config = new RequestConfig();
               config.setFilter(filter);
               CustomerProfileDocument doc[] = (CustomerProfileDocument[]) ds.invoke("getCustomerProfile",param

        // Show Customer Data
              System.out.println("══════════════ Customer ═════════════");
              Customer customer = doc[0].getCustomerProfile().getCustomerArray(0);
              System.out.println("Connected to Liquid Data 8.2 : CustomerProfile Data Service ...");

// Get the order item
    //     OrderLine myItem = customer.getOrders().getOrderArray(0).getOrderLineArray(4);
    //     myItem.delete();               // Delete the order item
    //     ds.submit(doc);                // Submit delete order item
    //     System.out.println("Change Submitted");

// Show Order Data
              System.out.println("═══════════════ Orders ═════════════");
              Order[] order = customer.getOrders().getOrderArray();
              for (int x=0; x<order.length; x++) {
                  System.out.println("     Order # " + order[x].getOrderId() +
                                  "      Date " + order[x].getOrderDate() +
                                  "      Total $' + order[x].getTotalOrderAmount() );
                  OrderLine[] orderline = order[x].getOrderLineArray();
```

8. Click the DataServiceClient.java file's Start icon (or press Ctrl + F5).

9. Use the Mediator API to view the results in the Output window and/or a standalone Java environment. The return results should be similar to those displayed in Figure 11-2.

Figure 11-2 Filtered Data Results



```
Trying to create process and attach to 1900...
C:\bea\jrockit81sp4_142_05\bin\javaw.exe -Xdebug -Xnoagent -Djava.compiler=NONE -Xrunjdwp:transp
Process started
Attached successfully.
==================== Data Service Client ====================
Connected to Liquid Data 8.2 : CustomerProfile Data Service ...
====================== Customer ====================
Customer Name : Pierce, Britt
======================= Orders ======================
     Order # ORDER_3_2     Date 2002-01-02     Total $1283.65
          Product # APPA_BA_1          Price $99.95          Quantity: 1
          Product # APPA_BA_1          Price $325.95          Quantity: 1
          Product # APPA_BA_3          Price $850.95          Quantity: 1
     Order # ORDER_3_3     Date 2002-02-17     Total $1679.65
          Product # APPA_BA_1          Price $325.95          Quantity: 1
          Product # APPA_BA_3          Price $850.95          Quantity: 1
          Product # APPA_BA_4          Price $495.95          Quantity: 1
     Order # ORDER_3_4     Date 2002-04-05     Total $1944.65
          Product # APPA_BA_3          Price $850.95          Quantity: 1
          Product # APPA_BA_4          Price $495.95          Quantity: 1
          Product # APPA_BA_5          Price $590.95          Quantity: 1
     Order # ORDER_3_5     Date 2002-05-21     Total $1106.65
          Product # APPA_BA_4          Price $495.95          Quantity: 1
          Product # APPA_BA_5          Price $590.95          Quantity: 1
          Product # APPA_WN_1          Price $12.95          Quantity: 1
Debugging Finished
```

11.2 Sorting Data Service Results

With the FilterXQuery `class sortfilter.addOrderBy()` method, you can specify criteria for organizing the data service return results. For example, to sort the order amount results in ascending order, you would use a sort condition similar to the following:

```
("CustomerProfile/customer/orders/order","total_order_amount",

FilterXQuery.ASCENDING);
```

Objectives

In this exercise, you will:

- Add a sort condition.
- View the results using the Mediator API.

Instructions

1. Open the `DataServiceClient.java` file.
2. Create a sort instance of the FilterXQuery, by adding the following code before the //Apply Filter section:

```
// Create a sort

FilterXQuery sortfilter = new FilterXQuery();
```

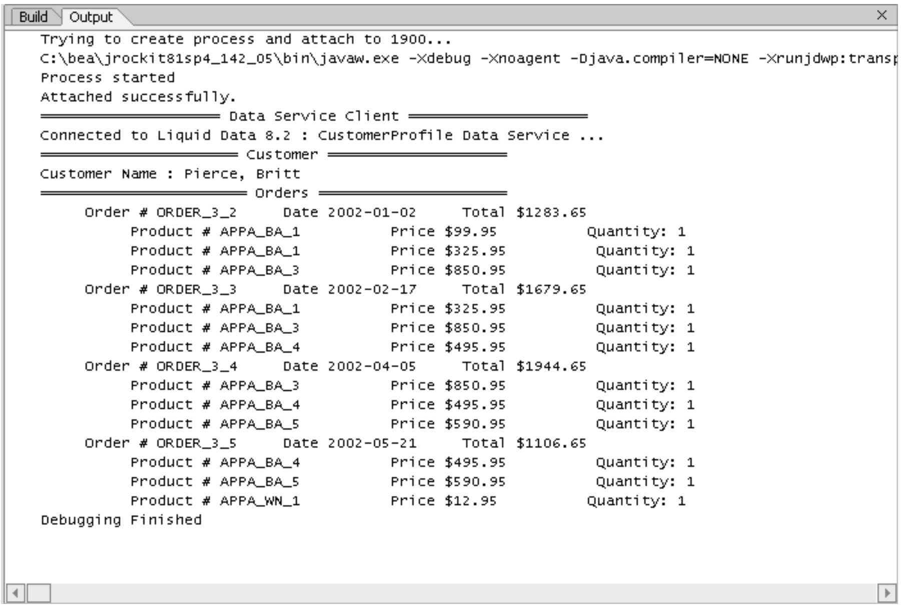
3. Add a sort condition, using the `addOrderBy()` method, to sort orders based on `total_order_amount (ascending)` as shown:

```
sortfilter.addOrderBy(

"CustomerProfile/customer/orders/order",

"total_order_amount",

FilterXQuery.ASCENDING);
```

4. Apply the sort filter to the data service by adding the following code:

```
// Apply the sort

filter.setOrderByList(sortfilter.getOrderByList());
```

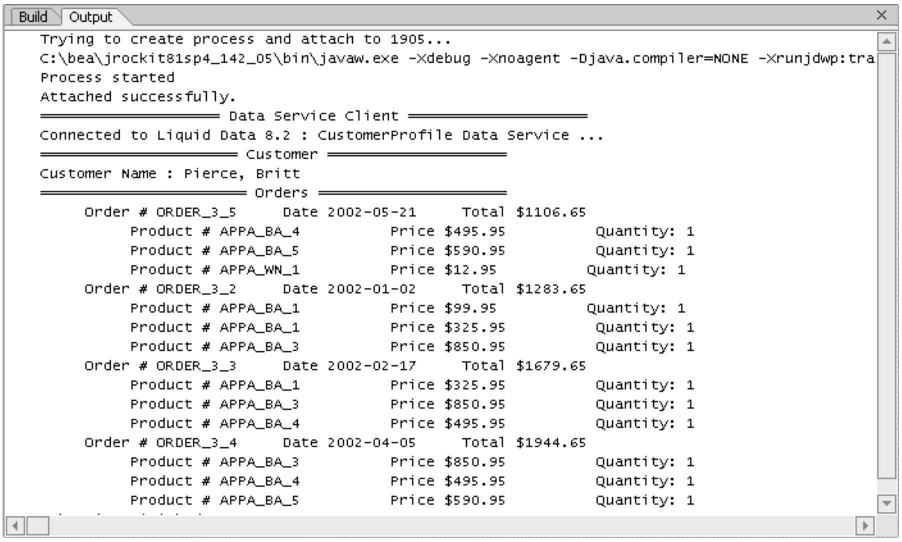
Figure 11-3 Sort Code



```
Trying to create process and attach to 1900...
C:\bea\jrockit81sp4_142_05\bin\javaw.exe -Xdebug -Xnoagent -Djava.compiler=NONE -Xrunjdwp:transp
Process started
Attached successfully.
==================== Data Service Client ====================
Connected to Liquid Data 8.2 : CustomerProfile Data Service ...
====================== Customer ====================
Customer Name : Pierce, Britt
======================= Orders ======================
     Order # ORDER_3_2     Date 2002-01-02     Total $1283.65
          Product # APPA_BA_1          Price $99.95          Quantity: 1
          Product # APPA_BA_1          Price $325.95          Quantity: 1
          Product # APPA_BA_3          Price $850.95          Quantity: 1
     Order # ORDER_3_3     Date 2002-02-17     Total $1679.65
          Product # APPA_BA_1          Price $325.95          Quantity: 1
          Product # APPA_BA_3          Price $850.95          Quantity: 1
          Product # APPA_BA_4          Price $495.95          Quantity: 1
     Order # ORDER_3_4     Date 2002-04-05     Total $1944.65
          Product # APPA_BA_3          Price $850.95          Quantity: 1
          Product # APPA_BA_4          Price $495.95          Quantity: 1
          Product # APPA_BA_5          Price $590.95          Quantity: 1
     Order # ORDER_3_5     Date 2002-05-21     Total $1106.65
          Product # APPA_BA_4          Price $495.95          Quantity: 1
          Product # APPA_BA_5          Price $590.95          Quantity: 1
          Product # APPA_WN_1          Price $12.95          Quantity: 1
Debugging Finished
```

5. Click the Start icon (or press Ctrl + F5) for the DataServiceClient.java file.
6. Use the Mediator API to view the results in the Output window and/or a standalone Java environment. The data results should be similar to those displayed in Figure 11-4.

Figure 11-4 Filtered and Sorted Data Results



```
Trying to create process and attach to 1905...
C:\bea\jrockit81sp4_142_05\bin\javaw.exe -Xdebug -Xnoagent -Djava.compiler=NONE -Xrunjdwp:tra
Process started
Attached successfully.
==================== Data Service Client ====================
Connected to Liquid Data 8.2 : CustomerProfile Data Service ...
====================== Customer ====================
Customer Name : Pierce, Britt
======================= Orders ====================
     Order # ORDER_3_5     Date 2002-05-21     Total $1106.65
          Product # APPA_BA_4          Price $495.95          Quantity: 1
          Product # APPA_BA_5          Price $590.95          Quantity: 1
          Product # APPA_WN_1          Price $12.95          Quantity: 1
     Order # ORDER_3_2     Date 2002-01-02     Total $1283.65
          Product # APPA_BA_1          Price $99.95          Quantity: 1
          Product # APPA_BA_1          Price $325.95          Quantity: 1
          Product # APPA_BA_3          Price $850.95          Quantity: 1
     Order # ORDER_3_3     Date 2002-02-17     Total $1679.65
          Product # APPA_BA_1          Price $325.95          Quantity: 1
          Product # APPA_BA_3          Price $850.95          Quantity: 1
          Product # APPA_BA_4          Price $495.95          Quantity: 1
     Order # ORDER_3_4     Date 2002-04-05     Total $1944.65
          Product # APPA_BA_3          Price $850.95          Quantity: 1
          Product # APPA_BA_4          Price $495.95          Quantity: 1
          Product # APPA_BA_5          Price $590.95          Quantity: 1
```

11.3 Truncating Data Service Results

The FilterXQuery class also provides the filter.setLimit() method, which lets you limit the number of return results. For example, to limit the return results to two line items, you would use a truncate condition similar to the following:

```
("CustomerProfile/customer/orders/order/order_line",”2”);
```

The filter.setLimit method is based on the following:

```
public void setLimit(java.lang.String appliesTo, String max)
```

Objectives

In this exercise, you will:

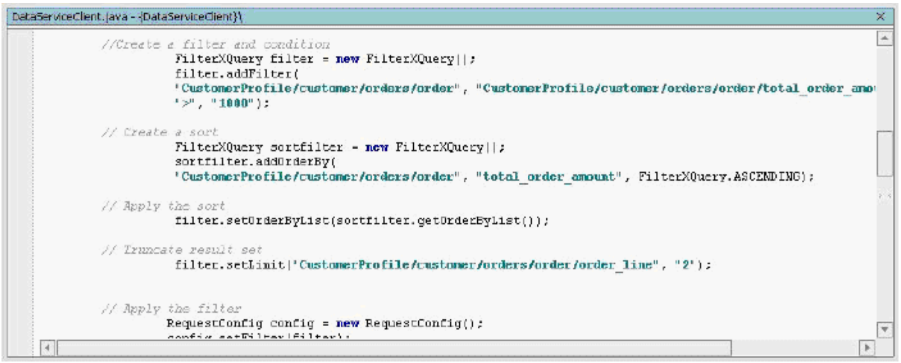
- Truncate the data result set.
- View the results using the Mediator API.

Instructions

1. Open the `DataServiceClient.java` file.
2. Add a truncate condition, using the `setLimit()` method to limit the result set to a maximum of two order lines for each order, as shown:

```
// Truncate result set

  filter.setLimit("CustomerProfile/customer/orders/order/order_line",”2”);
```

Figure 11-5 Truncate Code

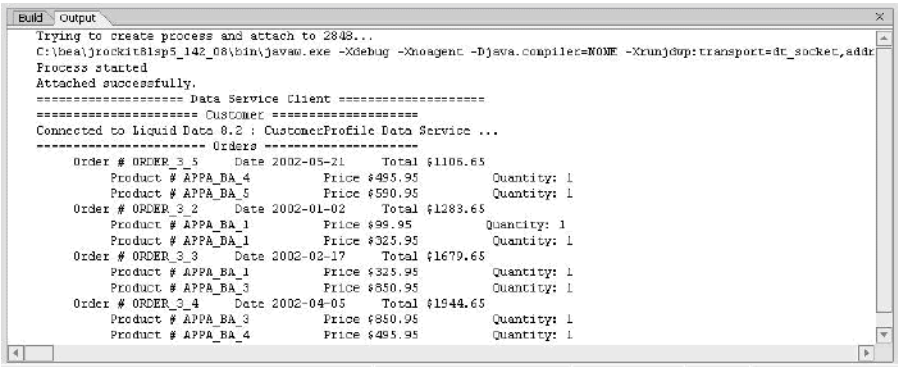


DataServiceClient.java - {DataServiceClient}\

```
        //Create a filter and condition
                FilterXQuery filter = new FilterXQuery();
                filter.addFilter(
                "CustomerProfile/customer/orders/order", "CustomerProfile/customer/orders/order/total_order_amo
                ">", "1000");

        // Create a sort
                FilterXQuery sortfilter = new FilterXQuery();
                sortfilter.addOrderBy(
                "CustomerProfile/customer/orders/order", "total_order_amount", FilterXQuery.ASCENDING);

        // Apply the sort
                filter.setOrderByList(sortfilter.getOrderByList());

        // Truncate result set
                filter.setLimit("CustomerProfile/customer/orders/order/order_line", "2");


        // Apply the filter
               RequestConfig config = new RequestConfig();
```

3. Click the Start icon (or press Ctrl + F5) for the DataServiceClient.java file.
4. Use the Mediator API to view the results in the Output window and/or a standalone Java environment. The data results should be similar to those displayed in Figure 11-6.

Figure 11-6 Truncated Result Set



```
Trying to create process and attach to 2848...
C:\bea\jrockit81sp5_142_08\bin\javaw.exe -Xdebug -Xnoagent -Djava.compiler=NONE -Xrunjdwp:transport=dt_socket,addr
Process started
Attached successfully.
==================== Data Service Client ====================
===================== Customer ====================
Connected to Liquid Data 8.2 : CustomerProfile Data Service ...
----------------------- Orders --------------------
     Order # ORDER_3_5     Date 2002-05-21     Total $1106.65
          Product # APPA_BA_4          Price $495.95          Quantity: 1
          Product # APPA_BA_5          Price $590.95          Quantity: 1
     Order # ORDER_3_2     Date 2002-01-02     Total $1283.65
          Product # APPA_BA_1          Price $99.95          Quantity: 1
          Product # APPA_BA_1          Price $325.95          Quantity: 1
     Order # ORDER_3_3     Date 2002-02-17     Total $1679.65
          Product # APPA_BA_1          Price $325.95          Quantity: 1
          Product # APPA_BA_3          Price $850.95          Quantity: 1
     Order # ORDER_3_4     Date 2002-04-05     Total $1944.65
          Product # APPA_BA_3          Price $850.95          Quantity: 1
          Product # APPA_BA_4          Price $495.95          Quantity: 1
```

Lesson Summary

In this lesson, you learned how to:

- Use the FilterXQuery class to filter, sort, and truncate data service results.
- Apply the FilterXQuery class to a data service, using the Mediator API or Data Service Control.

Consuming Data Services through JDBC/SQL

Data Services Platform JDBC driver gives JDBC clients read-only access to the information supplied by data services. With the Data Services Platform JDBC driver, ALDSP acts as a virtual database. The driver allows you to invoke data service functions from any JDBC client, from custom Java applications to database, and from reporting tools, including Crystal Reports.

Objectives

After completing this lesson, you will be able to:

- Access ALDSP via JDBC.
- Integrate a Crystal Report file, populated by ALDSP, into your Web application.
- Access ALDSP via Crystal Reports 11.

Overview

Data services built into ALDSP can be accessed using the Data Services Platform JDBC driver, which provides access to the ALDSP-enabled Server via JDBC APIs. With this functionality, JDBC clients—including business intelligence and reporting tools such as Business Objects and Crystal Reports—are granted read-only access to the information supplied by ALDSP services. The main features of the Data Services Platform JDBC driver are:

- Supports SQL-92 SELECT statements.

- Provides error handling; if an error is detected in SQL query, then the error will be reported along with an error code.
- Performs metadata validation; the translator checks SQL syntax and validates it against the data service schema.

When communicating with ALDSP via a JDBC/ODBC interface, standard SQL-92 query language is supported. The Data Services Platform JDBC driver implements components of the java.sql.* interface, as specified in JDK 1.4x.

Data Services Platform JDBC driver gives JDBC clients read-only access to the information supplied by data services. With the Data Services Platform JDBC driver, ALDSP acts as a virtual database. The driver allows you to invoke data service functions from any JDBC client, from custom Java applications to database, and from reporting tools, including Crystal Reports.

12.1 Running DBVisualizer

WebLogic Platform includes DBVisualizer, which is a third-party database tool designed to simplify database development and management.

Before you start:

- The Data Services Platform JDBC driver needs to be in your computer's CLASSPATH:

  `$BEA_HOME\weblogic81\liquiddata\lib\ldjdbc.jar`

- Similarly, the WebLogic JAR file needs to be in your computer's CLASSPATH:

  `$BEA_HOME\weblogic81\server\lib\weblogic.jar`

- The WebLogic Server needs to be running.
- Make sure that your Evaluation application is deployed correctly to WebLogic Server.

Objectives

In this exercise, you will:

- Create a database connection that enables DBVisualizer to access your Evaluation application as if it were a database.
- Use DBVisualizer to explore your Evaluation application.

Instructions

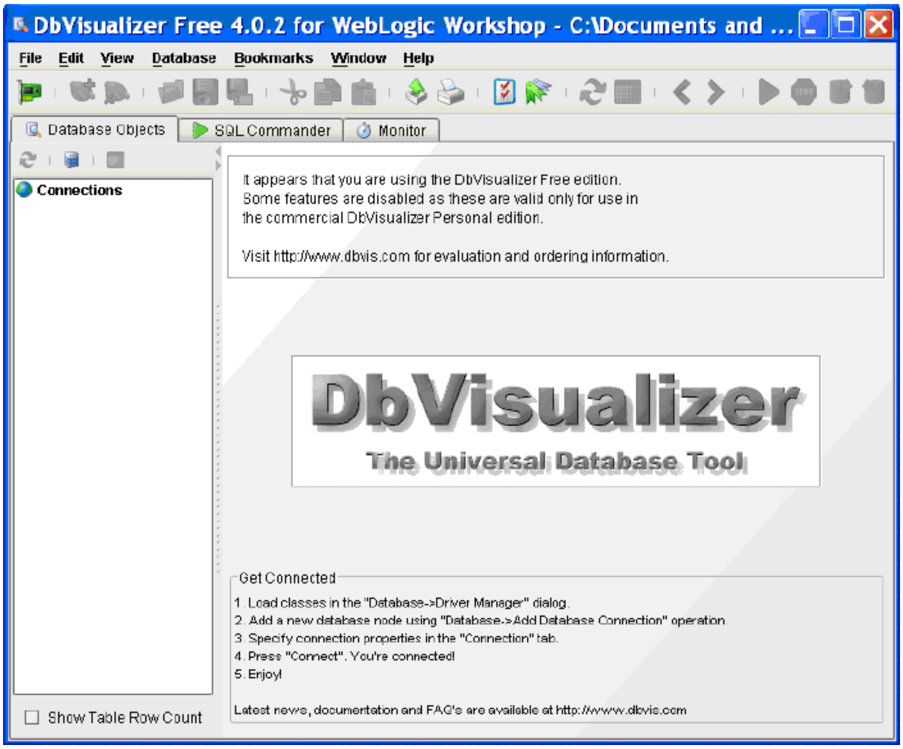
1. Publish your Evaluation data service functions for SQL use. For details see “Publishing Data Service Functions for SQL Use” in the Designing Data Services chapter of the *Data Services Developer’s Guide*.

   `http://edocs.bea.com/aldsp/docs25/datasrvc/xds.html#wp1111244`

2. Build your application.

3. Choose Start → Programs → BEA WebLogic Platform8.1→ Other Development Tools → DBVisualizer. The DBVisualizer tool opens.

Figure 12-1 DBVisualizer Interface


DbVisualizer Free 4.0.2 for WebLogic Workshop - C:\Documents and ...
File Edit View Database Bookmarks Window Help
Database Objects
SQL Commander
Monitor
Connections
Show Table Row Count
DbVisualizer
The Universal Database Tool

It appears that you are using the DbVisualizer Free edition.
Some features are disabled as these are valid only for use in
the commercial DbVisualizer Personal edition.

Visit http://www.dbvis.com for evaluation and ordering information.

Get Connected

1. Load classes in the "Database->Driver Manager" dialog.
2. Add a new database node using "Database->Add Database Connection" operation.
3. Specify connection properties in the "Connection" tab.
4. Press "Connect". You're connected!
5. Enjoy!

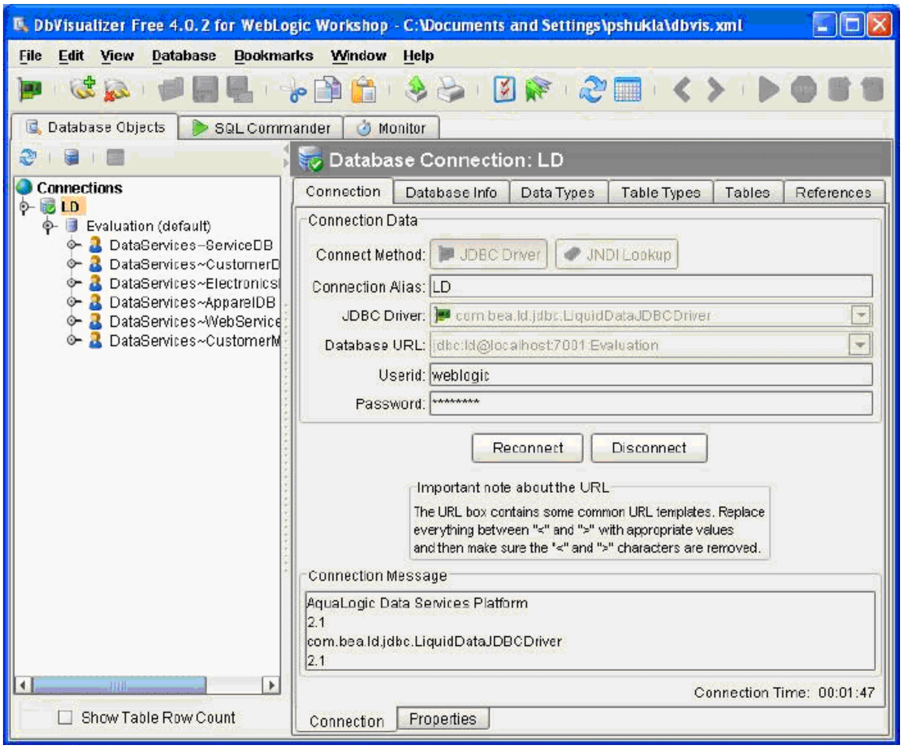
Latest news, documentation and FAQ's are available at http://www.dbvis.com

4. Choose Database → Add Database Connection.

5. Select the JDBC Driver tab from the Connection Data section.

6. Enter the following parameters:
   - Connection Alias: LD
   - JDBC Driver: com.bea.dsp.jdbc.DSPJDBCDriver
   - Database URL: jdbc:dsp@localhost:7001/Evaluation
   - Userid: weblogic
   - Password: weblogic
7. Click Connect.

Figure 12-2 New Database Connection Parameters



Connect Method: JDBC Driver

Connection Alias: LD

JDBC Driver: com.bea.ld.jdbc.LiquidDataJDBCDriver

Database URL: jdbc:ld@localhost:7001:Evaluation

Userid: weblogic

Password: ********

Important note about the URL

The URL box contains some common URL templates. Replace everything between "<" and ">" with appropriate values and then make sure the "<" and ">" characters are removed.

Connection Message

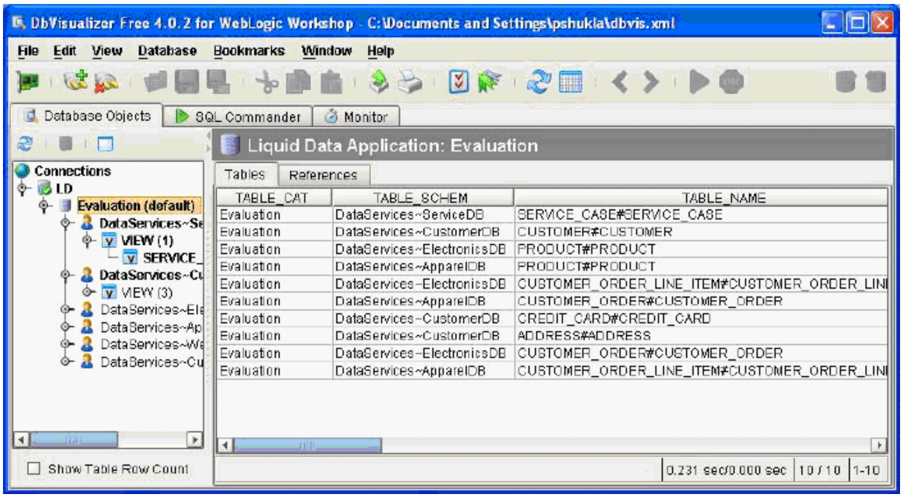
```
AquaLogic Data Services Platform
2.1
com.bea.ld.jdbc.LiquidDataJDBCDriver
2.1
```

Connection Time: 00:01:47

8. Use DBVisualizer to explore your ALDSP application as if it were a database. Data service projects display as database schemas. Functions within a project display as a database view; functions with parameters display as database functions.

9. Select a tab (Database Info, Data Types, Table Types, Tables, and References) to view that category of information for all data services within your application. For example, selecting the Tables tab displays each data service as a table.

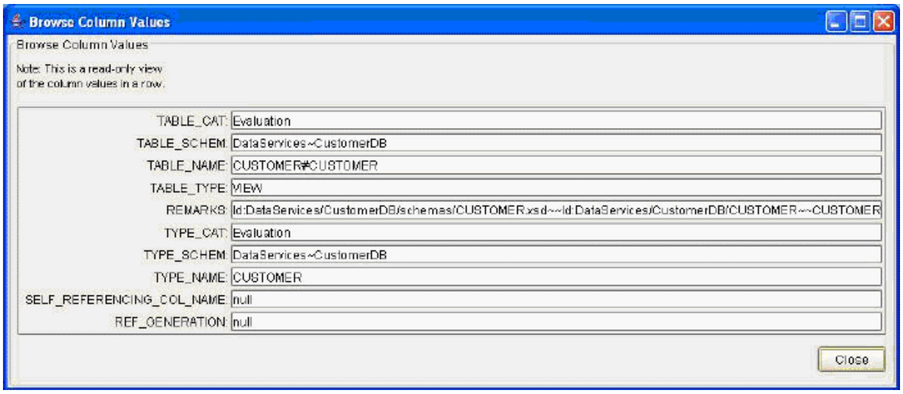
Figure 12-3 Tables



TABLE_CAT	TABLE_SCHEM	TABLE_NAME
Evaluation	DataServices~ServiceDB	SERVICE_CASE#SERVICE_CASE
Evaluation	DataServices~CustomerDB	CUSTOMER#CUSTOMER
Evaluation	DataServices~ElectronicsDB	PRODUCT#PRODUCT
Evaluation	DataServices~ApparelDB	PRODUCT#PRODUCT
Evaluation	DataServices~ElectronicsDB	CUSTOMER_ORDER_LINE_ITEM#CUSTOMER_ORDER_LINI
Evaluation	DataServices~ApparelDB	CUSTOMER_ORDER#CUSTOMER_ORDER
Evaluation	DataServices~CustomerDB	CREDIT_CARD#CREDIT_CARD
Evaluation	DataServices~CustomerDB	ADDRESS#ADDRESS
Evaluation	DataServices~ElectronicsDB	CUSTOMER_ORDER#CUSTOMER_ORDER
Evaluation	DataServices~ApparelDB	CUSTOMER_ORDER_LINE_ITEM#CUSTOMER_ORDER_LINI

10. Double-click an element to view the values for a specific data service. For example, double-clicking the DataServices~CustomerDB element from the Table Schema column displays that data services values.

Figure 12-4 Table Column Values



Browse Column Values

Note: This is a read-only view of the column values in a row.

TABLE_CAT:	Evaluation
TABLE_SCHEM:	DataServices~CustomerDB
TABLE_NAME:	CUSTOMER#CUSTOMER
TABLE_TYPE:	VIEW
REMARKS:	ld:DataServices/CustomerDB/schemas/CUSTOMER.xsd~~ld:DataServices/CustomerDB/CUSTOMER~~CUSTOMER
TYPE_CAT:	Evaluation
TYPE_SCHEM:	DataServices~CustomerDB
TYPE_NAME:	CUSTOMER
SELF_REFERENCING_COL_NAME:	null
REF_GENERATION:	null

12.2 Integrating Crystal Reports and Data Services Platform

The Data Services Platform JDBC driver makes data services accessible from business intelligence and reporting tools, such as Crystal Reports, Business Objects, Cognos, and so on. In this exercise, you will learn how to use the Date Service Platform JDBC driver in conjunction with Crystal Reports. (For ODBC applications, you can use JDBC to ODBC Bridge Drivers provided by vendors such as OpenLink, available as of this writing at http://www.openlinksw.com.)

Objectives

In this exercise, you will:

- Install Crystal Reports View in a Web application.
- Import a saved Crystal Report file and JSP into the Web application.
- View the report from the Web application.

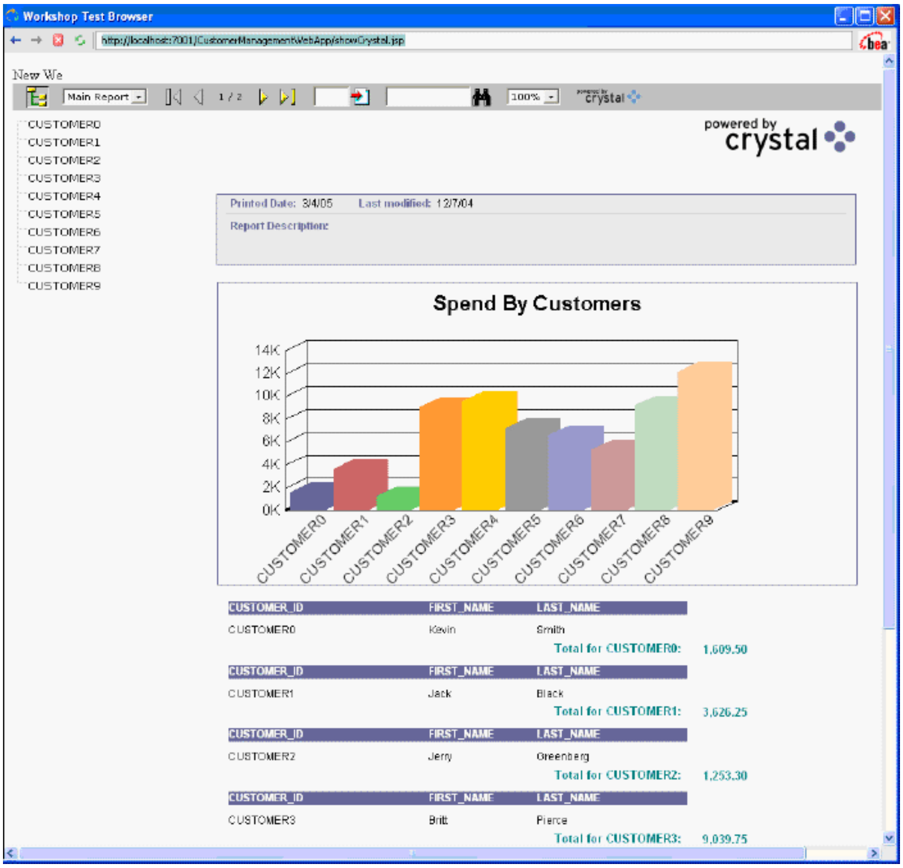
Instructions

1. Install Crystal Reports Viewer in the CustomerManagementWebApp by completing the following steps:
   a. Right-click CustomerManagementWebApp.
   b. Choose Install → Crystal Reports.
2. Import a saved Crystal Reports file and JSP that displays the report by completing the following steps:
   a. Right-click CustomerManagementWebApp.
   b. Choose Import.
   c. Navigate to <beahome>\weblogic81\samples\LiquidData\EvalGuide and select the SpendByCustomers.rpt and showCrystal.jsp files:
   d. Click Import. You should see show`Crystal.jsp` and `SpendByCustomers.rpt` files within CustomerManagementWebApp.
   e. Right-click the CustomerPageFlow folder.

  f. Choose Import.
  g. Select index.jsp, located in <beahome>\weblogic81\samples\LiquidData\EvalGuide.
  h. Click Import and choose Yes when asked if you want to overwrite the existing index.jsp file.

3. Open `CustomerPageFlowController.jpf`, located in CustomerManagementWebApp\CustomerPageFlow.
4. Click the Start icon (or press Ctrl + F5) to run Workshop Test Browser.
5. In Workshop Test Browser, click Customer Report to test the report. The first invocation may take time to display.

Figure 12-5 Crystal Report



Printed Date: 3/4/05 Last modified: 12/7/04

Report Description:

CUSTOMER_ID	FIRST_NAME	LAST_NAME	
CUSTOMER0	Kevin	Smith	
		Total for CUSTOMER0:	1,609.50

CUSTOMER_ID	FIRST_NAME	LAST_NAME	
CUSTOMER1	Jack	Black	
		Total for CUSTOMER1:	3,626.25

CUSTOMER_ID	FIRST_NAME	LAST_NAME	
CUSTOMER2	Jerry	Greenberg	
		Total for CUSTOMER2:	1,253.30

CUSTOMER_ID	FIRST_NAME	LAST_NAME	
CUSTOMER3	Britt	Pierce	
		Total for CUSTOMER3:	9,039.75

12.3 (Optional) Configuring JDBC Access through Crystal Reports

Crystal Reports 11 comes with a direct JDBC interface, which can be used to interact with the Data Services Platform JDBC driver.

Objectives

In this exercise, you will:

- Install Crystal Reports software, JDBC driver, and Java server files.
- Add environment variables.
- Create a new JDBC data source in Crystal Reports.

Instructions

1. Install the Crystal Reports software, per the vendor's installation instructions.
2. Add the JAVA_HOME as an environment variable. For example:

   ```
   JAVA_HOME=C:\j2sdk1.4.2_06
   ```

   where:

   ```
   C:\j2sdk1.4.2_06
   ```

   identifies the Java SDK location on your computer.
3. Make sure that the `jvm.dll` is in the path variable for your computer. For example:

   ```
   <$BEA_HOME>\jdk142_04\jre\bin\server
   ```

4. Locate the Crystal Reports configuration file (`CRConfig.xml`). By default it is located on your Windows system in the following directory:

   ```
   Program Files/Common Files/Business Objects/3.0/java
   ```

5. Make the following changes to the file:

- In the <Classpath> element add the location of `ldjdbc.jar` and weblogic.jar to the classpath element. For example:

  ```
  C:\81sp5sql\weblogic81\server\lib\weblogic.jar;
  C:\81sp5sql\weblogic81\liquiddata\lib\ldjdbc.jar;
  ```

- In the < JDBCURL> element to point to the application that you want to connect to. For example:

  ```
  jdbc:dsp@localhost:7001/Evaluation
  ```

- In the <JDBCClassName> element point to the ALDSP JDBC driver class name:

  ```
  com.bea.dsp.jdbc.Driver.DSPJDBCDriver
  ```

- Set the <JDBCUserName> element to the user. For the Evaluation sample application the user is:

  ```
  weblogic
  ```

- Set the GenericJDBCDriver <Option> element to Yes.
- Change the <DatabaseStructure> element from the default:

  ```
  catalogs,tables
  ```

  to:

  ```
  catalogs,schemas,tables
  ```

- Set the <LogonStyle> element to:

  ```
  Standard
  ```

6. Create a new connection to a JDBC data source in Crystal Reports:

- Select JDBC as the connection type in the Connection Standard Report Creation wizard.
- Set the JDBC Driver to:

  ```
  com.bea.dsp.jdbc.Driver.DSPJDBCDriver
  ```

- Set the URL string to:

  ```
  jdbc:dsp@localhost:7001/Evaluation
  ```

- Provide a user name and password. For the Evaluation application that would be weblogic and weblogic.

7. Login to Crystal Reports. Once authenticated, Crystal Reports will display a view of the Evaluation application.

Lesson Summary

In this lesson, you learned how to:

- Access ALDSP via JDBC.
- Integrate a Crystal Reports file, populated by ALDSP, into your Web application.
- Access ALDSP via Crystal Reports 11.

Consuming Data via Streaming API

Streaming API allows developers to retrieve Aqualogic Data Services Platform (ALDSP) results in a streaming fashion.

Objectives

After completing this lesson, you will be able to:

- Stream results returned from AquaLogic Data Services Platform into a flat file.
- Test the results.

Overview

There are situations where you need to extract large amounts of data from operational systems using ALDSP. For those cases, ALDSP provides a data streaming API. Large data sets can be retrieved to application in a streaming fashion or be streamed directly to a file on server. All security enforcements previously defined will still be relevant in case of the streaming API.

When working with streaming API keep the following things in mind:

- The ability to get results as streams will be only available on the Server; there will not be any client-server support for this API.
- Only the Generic Data Service Interface is available for getting streaming results.

13.1 Stream results into a flat file

Objectives

In this exercise, you will:

- Create a new function that streams CustomerProfile information into a flat file.
- Import a new jsp file to access a streaming function.
- Test streaming data into a file.

Instructions

1. Import new index page into your application
   a. Right-click CustomerPageFlow located in CustomerManagementWebApp.
   b. Choose Import.
   c. Navigate to <beahome>\weblogic81\samples\LiquidData\EvalGuide\Streaming.
   d. Select `index.jsp` as the page to be imported.
   e. Click on Import button.
   f. Open `index.jsp` in the streaming folder and verify that you have a new link called "Export All Data".
2. Insert streaming function into your page flow
   a. Open `CustomerPageFlowController.jpf` located in CustomerManagementWebApp\ CustomerPageFlow
   b. Go to Source View.
   c. Add two additional methods into the page flow.
   d. Open `Streaming.txt` file located in <beahome>\weblogic81\samples\LiquidData\EvalGuide\Streaming.
   e. Copy and paste both functions found in `Streaming.txt` file immediately after method `submitChanges()` in the `CustomerPageFlowController.jpf` java page flow.

f. Press four times the key combination of Alt + Enter keys to import missing packages or type the following in import section of page flow:

```
import com.bea.ld.dsmediator.client.StreamingDataService;

import javax.naming.InitialContext;

import javax.naming.NamingException;

import com.bea.ld.dsmediator.client.DataServiceFactory;

import weblogic.jndi.Environment;
```

Note: If your application name is different from "Evaluation", locate "Evaluation" in newStreamingDataService method and rename it to reflect the name of your application.

g. Save your changes.

3. Start your `CustomerPageFlowController.jpf`
4. Once the application is started, click the Export All Data link
5. Verify that data is exported successfully by opening `customerexport.txt`, located in:

`<BEAHOME>\weblogic81\samples\domains\ldplatform`

13.2 Consume data in streaming fashion

Objectives

In this exercise, you will:

- Import a new version of CustomerPageFlow.
- Instantiate a new Streaming Data Service.
- Retrieve results into XMLInputStream object by calling getCustomerProfile function.
- Test fetching data from ALDSP in a streaming fashion.

Instructions

1. Import a new folder into your application
   a. Right-click CustomerManagementWebApp located in your Evaluation application.
   b. Choose Import.

c. Navigate to <beahome>\weblogic81\samples\LiquidData\EvalGuide.

d. Select CustomerPageFlowStream folder to be imported.

e. Click Import.

f. Open `CustomerPageFlowController.jpf` file in Source View.

g. Locate stream method and the following comments:

```
//instantiate and initialize your streaming data service here
```

h. Add the following code:

```
com.bea.dsp.dsmediator.client.StreamingDataService sds = null;

    //instantiate and initialize your streaming data service here

    sds =
com.bea.dsp.dsmediator.client.DataServiceFactory.newStreamingDataService(g
etInitialContext(), "Evaluation",
"ld:DataServices/CustomerManagement/CustomerProfile");
```

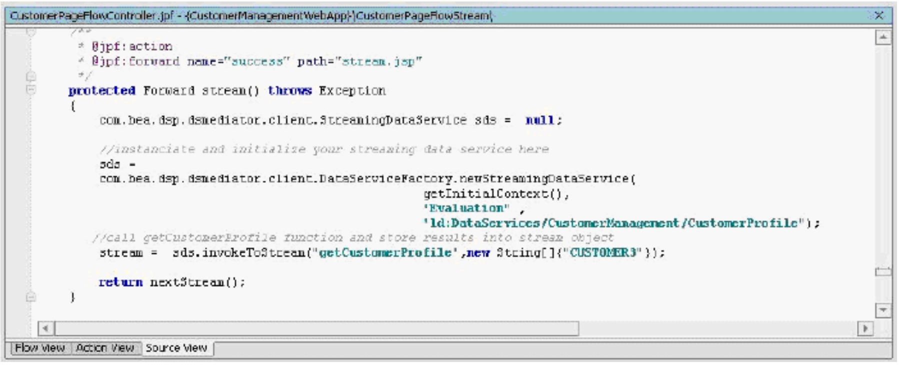
i. The DataServiceFactory class contains a method to create a streaming data service.

j. Replace stream = null with following code:

```
stream = sds.invoke("getCustomerProfile", new String[]{"CUSTOMER3"});
```

For reference, your code should look similar to that shown below:

Figure 13-1 Instantiating and Initializing Streaming Data

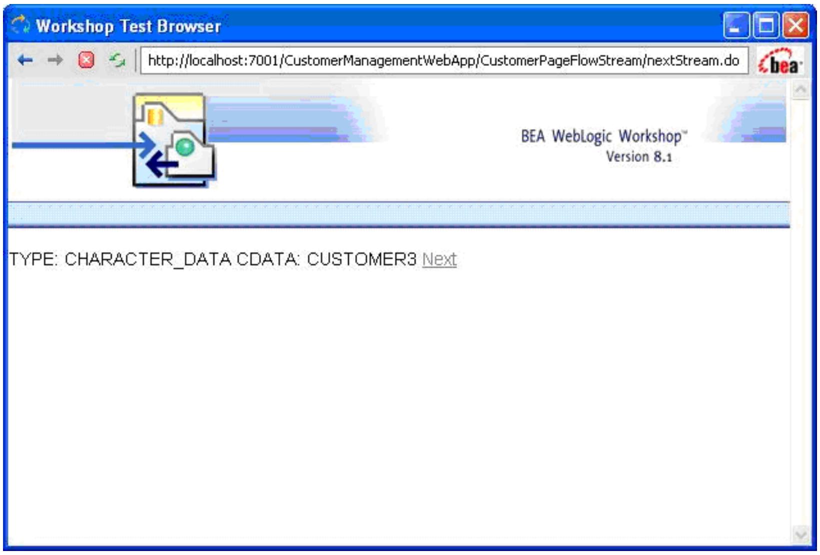


CustomerPageFlowController.jpf - {CustomerManagementWebApp}\CustomerPageFlowStream\

```
    /**
     * @jpf:action
     * @jpf:forward name="success" path="stream.jsp"
     */
    protected Forward stream() throws Exception
    {
        com.bea.dsp.dsmediator.client.StreamingDataService sds =  null;

        //instanciate and initialize your streaming data service here
        sds =
        com.bea.dsp.dsmediator.client.DataServiceFactory.newStreamingDataService(
                                                  getInitialContext(),
                                                  "Evaluation" ,
                                                  "ld:DataServices/CustomerManagement/CustomerProfile");
       //call getCustomerProfile function and store results into stream object
        stream =  sds.invokeToStream("getCustomerProfile",new String[]{"CUSTOMER3"});

        return nextStream();
    }
```

k. Test running your `CustomerPageFlowController.jpf`. You can use CUSTOMER3 as a parameter to retrieve results. This time, data is fetched in streaming fashion as shown in Figure 13-2.

Figure 13-2 Data in Streaming Format


Workshop Test Browser
http://localhost:7001/CustomerManagementWebApp/CustomerPageFlowStream/nextStream.do
bea
BEA WebLogic Workshop™
Version 8.1

TYPE: CHARACTER_DATA CDATA: CUSTOMER3 Next

Lesson Summary

In this lesson, you learned to:

- Stream results returned from AquaLogic Data Services Platform into a flat file.
- Test the results.

Managing Data Service Metadata

ALDSP uses a set of descriptors (or metadata) to provide information about data services. The metadata describes the data services: what information they provide and where the information derives from (that is, its lineage). In addition to documenting services for potential consumers, metadata helps administrators determine what services are affected when inevitable changes occur in the data source layer. If a database changes, you can easily tell which data services are affected by the change.

Objectives

After completing this lesson, you will be able to:

- Synchronize physical data service metadata with changes made to the physical data source.
- Analyze impacts and dependencies.
- Create custom metadata for a logical data service.

Overview

ALDSP metadata information is stored as annotations at the data service and function levels. The metadata is openly structured as XML fragments for easy export and import. At deployment time, the metadata is incorporated into a compiled data service, and then deployed as part of the data service application in WebLogic Server.

Stored metadata includes:

Physical data service metadata:

- Relational data source, type, and version
- Column names, native data types, size, and scale
- XML schema types
- Web service WSDL URI

User-defined metadata:

- Description
- Custom properties at the data service level
- Custom properties at the function level
- Relationships created through data modeling

The Data Services Platform Console lets you access metadata stored within the ALDSP metadata repository. The ALDSP Console supports the following functionality:

- Searching the metadata repository
- Exploring where and how a given data service or function is consumed
- Analyzing data service lineage and dependencies (all data service objects dependent on a given data service)

Imported physical data service metadata can be re-synchronized to capture changes at the data source.

14.1 Defining Customized Metadata for a Logical Data Service

There may be times when you need to modify the generated metadata descriptions to provide more detailed information to others who will be working with the data service.

Objectives

In this exercise, you will:

- Create customized metadata for the CustomerProfile logical data service, at both the data service and function levels.
- Build the DataServices project to enable persistence of the new metadata.

Instructions

1. Add customized metadata at the data service level, by completing the following steps:

   a. Open `CustomerProfile.ds` in Design View. The file is located in the DataServices\CustomerManagement.

   b. Click the data service header to open the Property Editor at the data service level. (If the Property Editor is not open, choose View → Property Editor, or press Alt + 6.)

   c. In Property Editor, click the Description field, located in the General section. This activates the Description field.

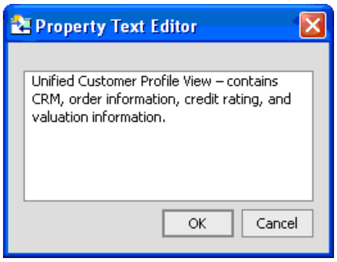
   d. Click the "..." icon for the Description field. The Property Text Editor opens.

   e. In Property Text Editor, enter the following text:

   f. Unified Customer Profile View – contains CRM, order information, credit rating, and valuation information.

   g. Click OK. The specified text is added to the Description field.

Figure 14-1 Property Text Editor



Unified Customer Profile View – contains CRM, order information, credit rating, and valuation information.

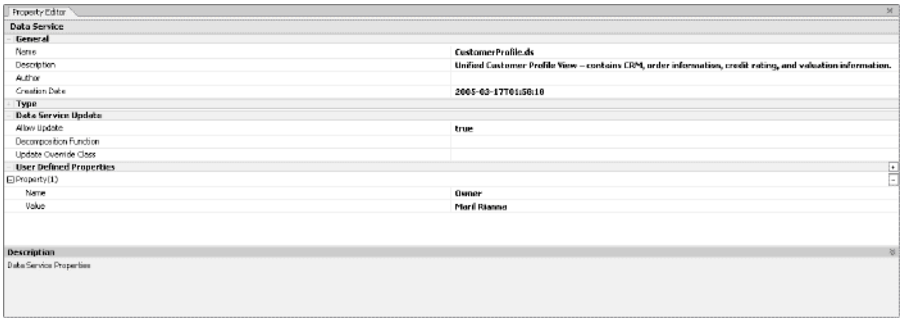
   h. In Property Editor, click the + icon for the User-Defined Properties section.

   i. Click the + icon for the Property(1) field. This activates the Property(1) field.

   j. Add a user-defined property, using the following values:

      - Name = Owner
      - Value = <your name>

Figure 14-2 User-Defined Property for a Logical Data Service



Property Editor

Data Service	
General	
Name	CustomerProfile.ds
Description	Unified Customer Profile View – contains CRM, order information, credit rating, and valuation information.
Author	
Creation Date	2005-03-17T01:58:10
Type	
Data Service Update	
Allow Update	true
Decomposition Function	
Update Override Class	
User Defined Properties	
Property(1)	
Name	Owner
Value	Mori Rianna

Description

Data Service Properties

2. Add customized metadata at the function level, by completing the following steps:

   a. In Design View, click the `getCustomerProfile()` function arrow to open that function's Property Editor.

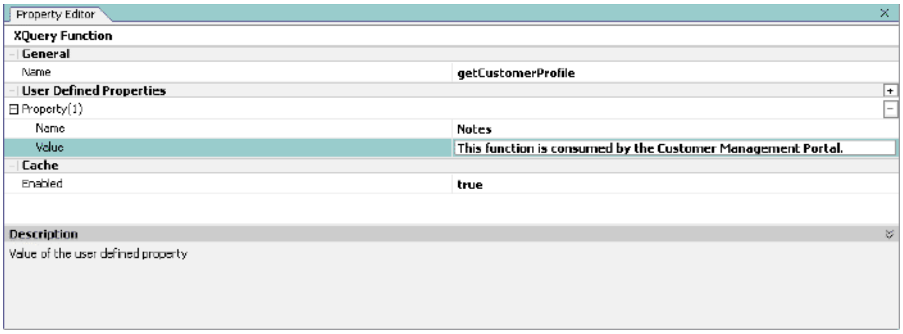
   Note: Do not click the function, which will open XQuery Editor View.

   b. In Property Editor, click the + icon, located in the User-Defined Properties section.

   c. Add a user-defined property, using the following values:

      - Name = Notes
      - Value = This function is consumed by the Customer Management Portal.

Figure 14-3 User-Defined Property for a Function



Property Editor

XQuery Function	
General	
Name	getCustomerProfile
User Defined Properties	
Property(1)	
Name	Notes
Value	This function is consumed by the Customer Management Portal.
Cache	
Enabled	true

Description

Value of the user defined property

3. Save the file.

4. Build the DataServices project.

14.2 Viewing Data Service Metadata Using the ALDSP Console

All data service metadata, whether automatically generated or user-defined, can be viewed using the ALDSP Console.

Objectives

In this exercise, you will:

- Use the ALDSP Console to view both generated and customized metadata.
- Use the console's Search feature to locate metadata for a specific data service.

Instructions

1. Open the ALDSP Console, typically located at `http://localhost:7001/ldconsole/`.

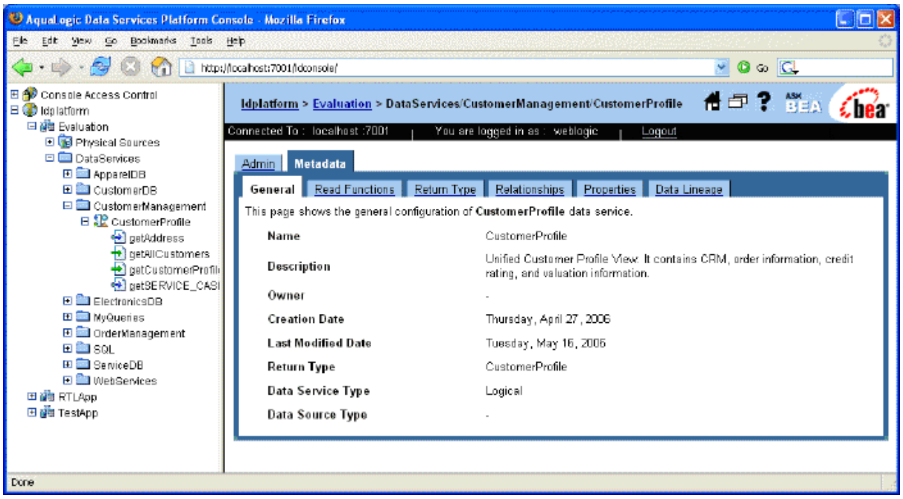
   Note: WebLogic Server must be running.

2. Log in using the following credentials:

- User = weblogic
- Password = weblogic

3. Open the CustomerProfile data service, located in ldplatform\Evaluation\DataServices\CustomerManagement using the left-hand menu.

Figure 14-4 ALDSP Console


AquaLogic Data Services Platform Console - Mozilla Firefox

Idplatform > Evaluation > DataServices/CustomerManagement/CustomerProfile

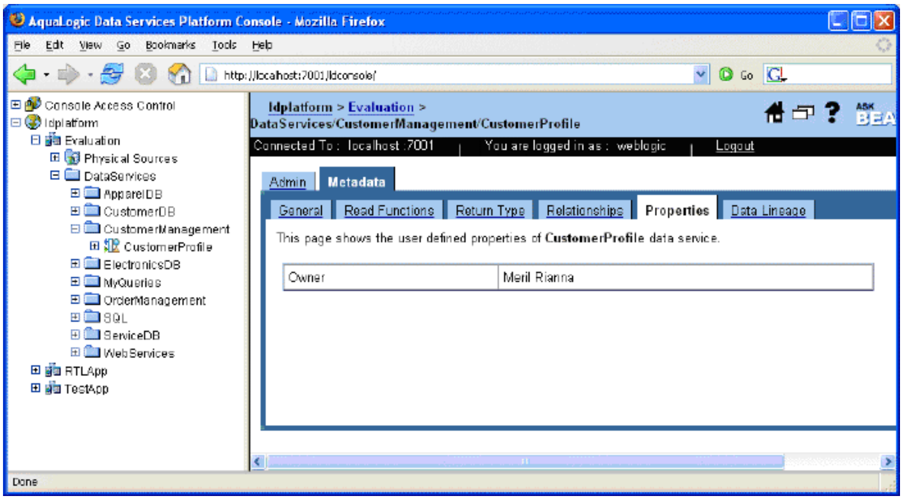
Connected To : localhost :7001 | You are logged in as : weblogic | Logout

This page shows the general configuration of **CustomerProfile** data service.

Name	CustomerProfile
Description	Unified Customer Profile View. It contains CRM, order information, credit rating, and valuation information.
Owner	-
Creation Date	Thursday, April 27, 2006
Last Modified Date	Tuesday, May 16, 2006
Return Type	CustomerProfile
Data Service Type	Logical
Data Source Type	-

4. Click the Properties tab and verify that user-defined properties for the data service display. The property should be similar to that displayed in Figure 14-5, except that it will be your name in the Value field.

Figure 14-5 Customer Profile Properties Metadata


AquaLogic Data Services Platform Console - Mozilla Firefox

Idplatform > Evaluation > DataServices/CustomerManagement/CustomerProfile

Connected To : localhost :7001 | You are logged in as : weblogic | Logout

This page shows the user defined properties of **CustomerProfile** data service.

Owner	Meril Rianna

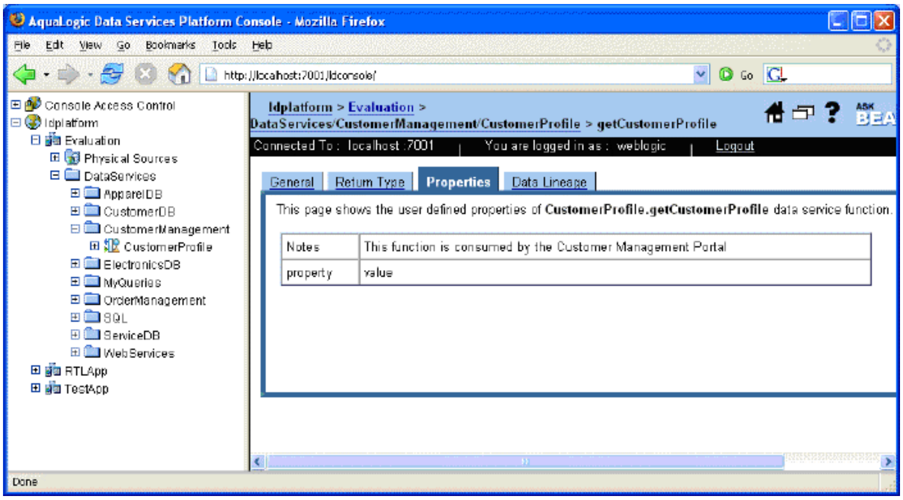
5. Explore the CustomerProfile data service metadata by completing the following steps:

   a. Select the Read Functions tab.

   b. Click `getCustomerProfile()`.

   c. Click the Properties tab. The Note that you created for `getCustomerProfile()` should be visible.

Figure 14-6 Metadata -- Read Function Properties


AquaLogic Data Services Platform Console - Mozilla Firefox
File Edit View Go Bookmarks Tools Help
http://localhost:7001/ldconsole/
Console Access Control
ldplatform
Evaluation
Physical Sources
DataServices
ApparelDB
CustomerDB
CustomerManagement
CustomerProfile
ElectronicsDB
MyQueries
OrderManagement
SQL
ServiceDB
WebServices
RTLApp
TestApp
ldplatform > Evaluation >
DataServices/CustomerManagement/CustomerProfile > getCustomerProfile
Connected To : localhost :7001
You are logged in as : weblogic
Logout
General
Return Type
Properties
Data Lineage
Done

This page shows the user defined properties of **CustomerProfile.getCustomerProfile** data service function.

Notes	This function is consumed by the Customer Management Portal
property	value

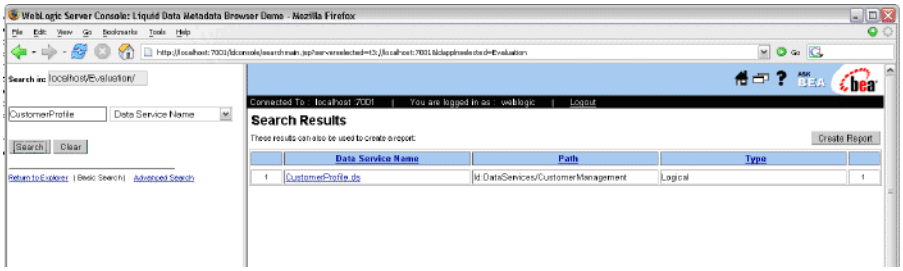
   d. (*Optional*) Select the Return Type, Relationships, Properties, and Where Used tabs to view other metadata.

6. Search the DataServices folder for metadata by completing the following steps:

   a. Right-click the Evaluation folder and click Search. (A search can be on data service name, function name, description, or return type.)

   b. Enter CustomerProfile in the Data Service Name search box and click Search. The data service name, path, and type of data service are displayed for the CustomerProfile data service. Clicking the data service name displays the Admin page for the data service.

Figure 14-7 Search Results


WebLogic Server Console: Liquid Data Metadata Browser Demo - Mozilla Firefox
File Edit View Go Bookmarks Tools Help
http://localhost:7001/ldconsole/searchmain.jsp?serverselected=t3://localhost:7001&ldappselected=Evaluation
Search in: localhost/Evaluation/
CustomerProfile
Data Service Name
Search
Clear
Return to Explorer | Basic Search | Advanced Search
ASK BEA
bea
Create Report

Connected To : localhost :7001 | You are logged in as : weblogic | Logout

Search Results

These results can also be used to create a report:

	Data Service Name	Path	Type	
1	CustomerProfile.ds	ld:DataServices/CustomerManagement	Logical	1

14.3 Synching a Data Service with Underlying Data Source Tables

Sometimes the underlying data source changes; for example, a new table is added to a database. For those inevitable situations, ALDSP provides an easy way to update a data service.

Objectives

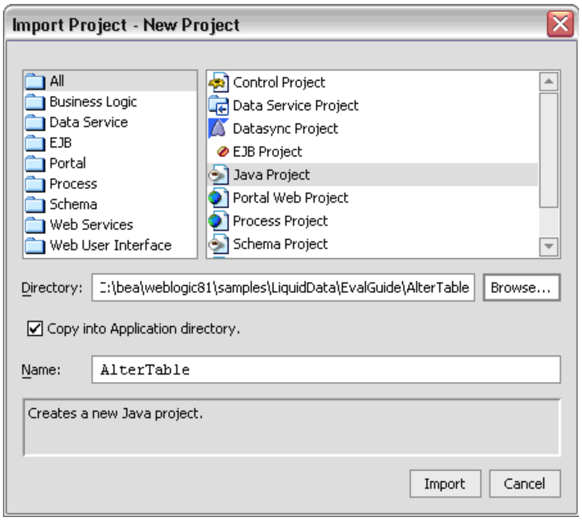
In this exercise, you will:

- Import a Java project that contains additional CUSTOMER_ORDER database columns.
- Synchronize the information in the Java project with the CUSTOMER_ORDER data service.
- Confirm the addition of a new element in the CUSTOMER_ORDER data service schema.

Instructions

1. In WebLogic Workshop, choose File → Import Project.
2. Select Java Project.
3. Navigate to <beahome>\weblogic81\samples\LiquidData\EvalGuide.
4. Select the AlterTable folder, click Open, and then click Import.

Figure 14-8 Importing Java Project



Directory: :\bea\weblogic81\samples\LiquidData\EvalGuide\AlterTable

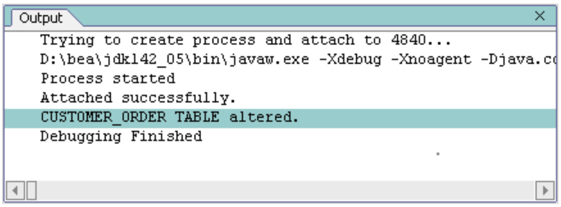
☑ Copy into Application directory.

Name: AlterTable

Creates a new Java project.

5. Open `AlterTable.java`. (The file is located in the AlterTable project folder).
6. Click the Start icon, and then click OK when a Confirmation message displays. Compiling the file adds a new column to the CUSTOMER_ORDER table.
7. Open the Output window and confirm that you see the CUSTOMER_ORDER_TABLE altered message.

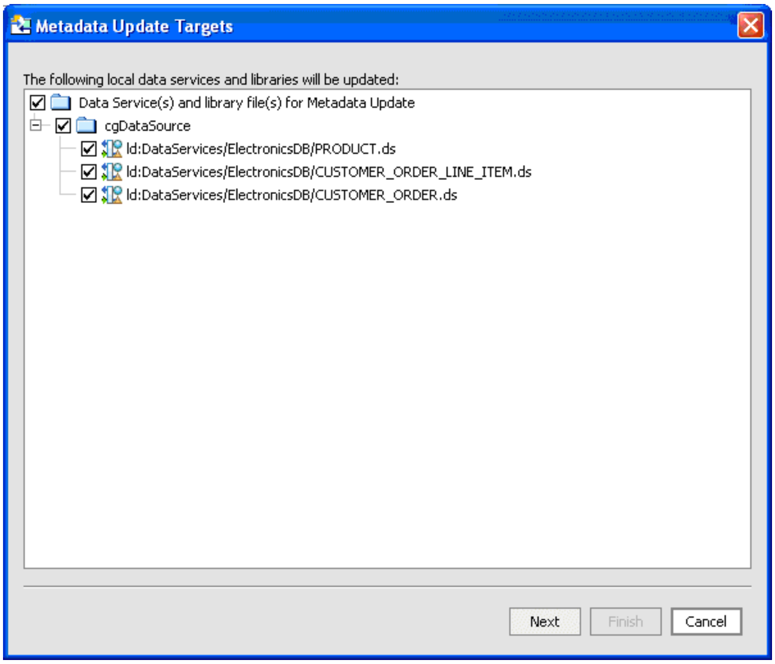
Figure 14-9 Altered Table Message



```
Trying to create process and attach to 4840...
D:\bea\jdk142_05\bin\javaw.exe -Xdebug -Xnoagent -Djava.c
Process started
Attached successfully.
CUSTOMER_ORDER TABLE altered.
Debugging Finished
```

8. Right-click the ElectronicsDB folder, located in the DataServices project folder.
9. Select Update Source Metadata. The Metadata Update Targets wizard opens, displaying a list of all new columns.

Figure 14-10 Physical Data Sources

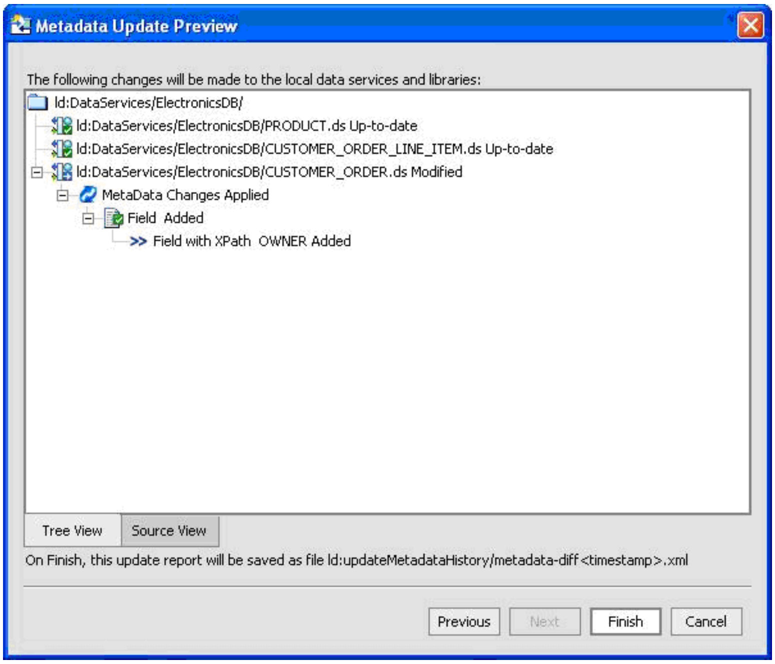


The following local data services and libraries will be updated:

- ☑ Data Service(s) and library file(s) for Metadata Update
  - ☑ cgDataSource
    - ☑ ld:DataServices/ElectronicsDB/PRODUCT.ds
    - ☑ ld:DataServices/ElectronicsDB/CUSTOMER_ORDER_LINE_ITEM.ds
    - ☑ ld:DataServices/ElectronicsDB/CUSTOMER_ORDER.ds

10. Click Next. The Metadata Update Preview dialog box opens, which provides details on the data to be synchronized.

Figure 14-11 Synchronization Preview



The following changes will be made to the local data services and libraries:

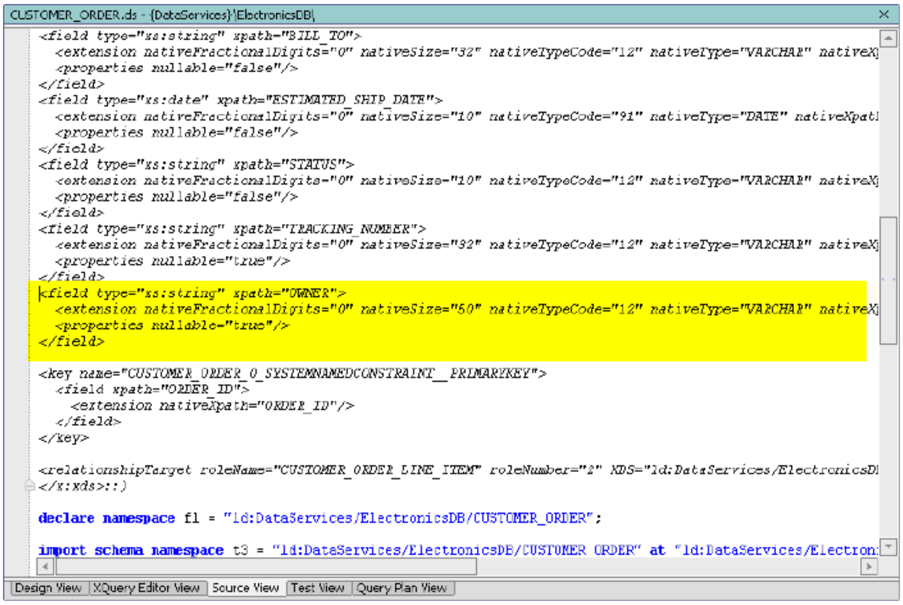
```
ld:DataServices/ElectronicsDB/
  ld:DataServices/ElectronicsDB/PRODUCT.ds Up-to-date
  ld:DataServices/ElectronicsDB/CUSTOMER_ORDER_LINE_ITEM.ds Up-to-date
  ld:DataServices/ElectronicsDB/CUSTOMER_ORDER.ds Modified
    MetaData Changes Applied
      Field  Added
        >> Field with XPath  OWNER Added
```

On Finish, this update report will be saved as file ld:updateMetadataHistory/metadata-diff<timestamp>.xml

11. Click Finish.
12. Open `CUSTOMER_ORDER.ds` in Source View. The file is located in the ElectronicsDB.
13. Expand the data service annotation, located on the first line of the file, to view the captured metadata for the relational data source (type, version, column names, native data types, size, scale, and XML schema types).
14. Scroll down until you locate the following code, which represents the customized metadata that you define in Exercise 14.1 Defining Customized Metadata for a Logical Data Service:

```
<field type="xs:string" xpath="OWNER">
  <extension nativeFractionalDigits="0" nativeSize="50"
nativeTypeCode="12" nativeType="VARCHAR" nativeXpath="OWNER"/>
  <properties nullable="true"/>
</field>
```

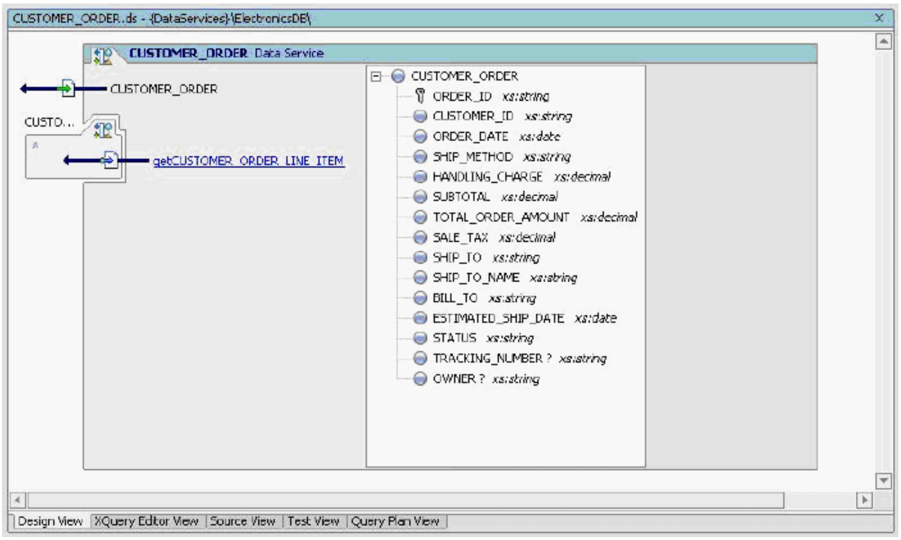
Figure 14-12 Source View of Updated Metadata



```
<field type="xs:string" xpath="BILL_TO">
  <extension nativeFractionalDigits="0" nativeSize="32" nativeTypeCode="12" nativeType="VARCHAR" nativeX
  <properties nullable="false"/>
</field>
<field type="xs:date" xpath="ESTIMATED_SHIP_DATE">
  <extension nativeFractionalDigits="0" nativeSize="10" nativeTypeCode="91" nativeType="DATE" nativeXpat
  <properties nullable="false"/>
</field>
<field type="xs:string" xpath="STATUS">
  <extension nativeFractionalDigits="0" nativeSize="10" nativeTypeCode="12" nativeType="VARCHAR" nativeX
  <properties nullable="false"/>
</field>
<field type="xs:string" xpath="TRACKING_NUMBER">
  <extension nativeFractionalDigits="0" nativeSize="32" nativeTypeCode="12" nativeType="VARCHAR" nativeX
  <properties nullable="true"/>
</field>
<field type="xs:string" xpath="OWNER">
  <extension nativeFractionalDigits="0" nativeSize="50" nativeTypeCode="12" nativeType="VARCHAR" nativeX
  <properties nullable="true"/>
</field>

<key name="CUSTOMER_ORDER_0_SYSTEMNAMEDCONSTRAINT__PRIMARYKEY">
  <field xpath="ORDER_ID">
    <extension nativeXpath="ORDER_ID"/>
  </field>
</key>

<relationshipTarget roleName="CUSTOMER_ORDER_LINE_ITEM" roleNumber="2" XDS="ld:DataServices/ElectronicsD
</x:xds>::)

declare namespace f1 = "ld:DataServices/ElectronicsDB/CUSTOMER_ORDER";

import schema namespace t3 = "ld:DataServices/ElectronicsDB/CUSTOMER_ORDER" at "ld:DataServices/Electron
```

15. Select the Design View tab, and verify that an Owner element exists in the XML type for the CUSTOMER_ORDER data service.

Figure 14-13 Design View


CUSTOMER_ORDER.ds - {DataServices}\ElectronicsDB\
CUSTOMER_ORDER Data Service
CUSTOMER_ORDER
CUSTO...
getCUSTOMER_ORDER_LINE_ITEM
Design View
XQuery Editor View
Source View
Test View
Query Plan View

```
CUSTOMER_ORDER
    ORDER_ID  xs:string
    CUSTOMER_ID  xs:string
    ORDER_DATE  xs:date
    SHIP_METHOD  xs:string
    HANDLING_CHARGE  xs:decimal
    SUBTOTAL  xs:decimal
    TOTAL_ORDER_AMOUNT  xs:decimal
    SALE_TAX  xs:decimal
    SHIP_TO  xs:string
    SHIP_TO_NAME  xs:string
    BILL_TO  xs:string
    ESTIMATED_SHIP_DATE  xs:date
    STATUS  xs:string
    TRACKING_NUMBER ?  xs:string
    OWNER ?  xs:string
```

16. Right-click the CUSTOMER_ORDER Data Service header and select Display Native Type. Confirm that there is a new element, called OWNER VARCHAR(50).

Lesson Summary

In this lesson, you learned how to:

- Synchronize physical data service metadata with changes made to the physical data source.
- Analyze impacts and dependencies.
- Create custom metadata for a logical data service.

Managing Data Service Caching

Caching enables the use of previously obtained results for queries that are repeatedly executed with the same parameters. This helps reduce processing time and enhance overall system performance.

Objectives

After completing this lesson, you will be able to:

- Use the ALDSP Console to configure a ALDSP cache.
- Enable the cache for a data service function and define its time-to-live (TTL).
- Check the database to verify whether a cache is used.
- Determine the performance impact of the cache, by checking the query response time.
- Disable caching.

Overview

When ALDSP executes a query, it returns to the client the data that resulted from the query execution. If ALDSP caching is enabled, then ALDSP saves its results into a query results cache the first time a query is executed. The next time the query is run with the same parameters, ALDSP checks the cache configuration and, if the results are not expired, quickly retrieves the results from the cache, rather than re-running the query. Using the previously obtained results for queries that are repeatedly executed with the same parameters reduces processing time and enhances overall system performance.

By default, the query results cache is disabled. Once enabled, you can configure the cache for individual stored queries as needed, specifying how long query results are stored in the cache before they expire (time out), and explicitly flushing the query cache.

In general, the results cache should be periodically refreshed to reflect data changes in the underlying data stores. The more dynamic the underlying data, the more frequently the cache should expire. For queries on static data (data that never changes), you can configure the results cache so that it never expires. For extremely dynamic data, you would never enable caching.

If the cache policy expires for a particular query, ALDSP automatically flushes the cache result on the next invocation. In the event of a Server shutdown, the contents of the results cache are retained. On the server restart, the Server resumes caching as before. On the first invocation of a cached query, ALDSP checks the results cache to determine whether the cached results for that query are valid or expired, and then proceeds accordingly.

15.1 Determining the Non-Cache Query Execution Time

To understand whether caching improves query execution time, you first need to know how long it takes to execute a non-cached query.

Objectives

In this exercise, you will:

- Execute a query function.
- Determine the query execution time.

Instructions

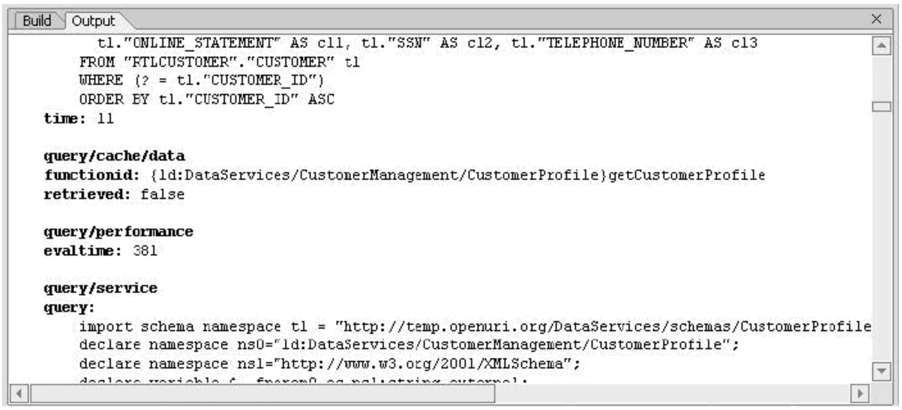
1. Open `CustomerProfile.ds` in Test View.
2. Select `getCustomerProfile(CustomerID)` from the function drop-down menu.
3. Enter CUSTOMER3 in the Parameter field.
4. Click Execute. The Output window displays the cache's execution time.

Note: Ensure that auditing is enabled in the ALDSP console, to view results in the Output window. For details about auditing, refer to the *Administrator's Guide*.

5. Open the Output window.

6. Search for query/performance evaltime for the value of query execution time.

Figure 15-1 Query Execution Time



```
      t1."ONLINE_STATEMENT" AS c11, t1."SSN" AS c12, t1."TELEPHONE_NUMBER" AS c13
    FROM "RTLCUSTOMER"."CUSTOMER" t1
    WHERE (? = t1."CUSTOMER_ID")
    ORDER BY t1."CUSTOMER_ID" ASC
time: 11

query/cache/data
functionid: {ld:DataServices/CustomerManagement/CustomerProfile}getCustomerProfile
retrieved: false

query/performance
evaltime: 381

query/service
query:
    import schema namespace t1 = "http://temp.openuri.org/DataServices/schemas/CustomerProfile
    declare namespace ns0="ld:DataServices/CustomerManagement/CustomerProfile";
    declare namespace ns1="http://www.w3.org/2001/XMLSchema";
```

15.2 Configuring a Caching Policy Through the ALDSP Console

By default, ALDSP results caching is disabled. You must explicitly enable caching. In this exercise, you will learn how to enable caching.

Objectives

In this exercise, you will:

- Enable caching at the application level.
- Enable caching at the function level.

Instructions

1. In the ALDSP Console (`http://localhost:7001/ldconsole/`), using the + icon, expand the ldplatform directory. (Note: If you click the ldplatform name, the Application List page opens. You do not want this page for this lesson.)
2. Enable caching at the application level, by completing the following steps:
   a. Click Evaluation. The ALDSP Console's General page opens.

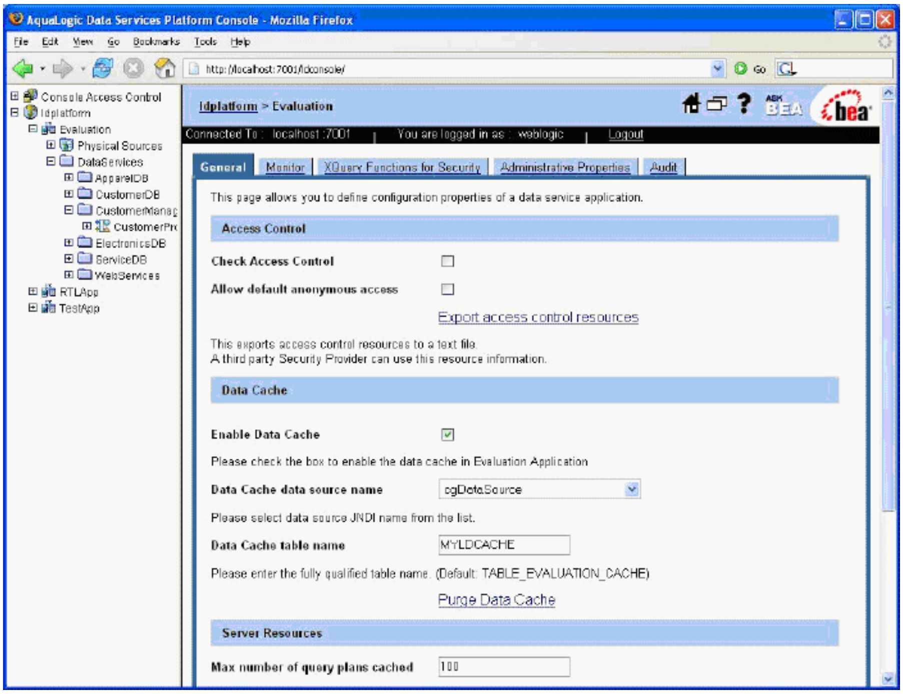
b. In the Data Cache section, select Enable Data Cache.

c. Select cgDataSource from the Data Cache data source name drop-down list.

d. Enter MYLDCACHE in the Data Cache table name field.

e. Click Apply.

Figure 15-2 ALDSP Console General Page


AquaLogic Data Services Platform Console - Mozilla Firefox
File Edit View Go Bookmarks Tools Help
http://localhost:7001/ldconsole/
Console Access Control
ldplatform
Evaluation
Physical Sources
DataServices
ApparelDB
CustomerDB
CustomerManag
CustomerPr
ElectronicsDB
ServiceDB
WebServices
RTLApp
TestApp
ldplatform > Evaluation
Connected To : localhost :7001 You are logged in as : weblogic Logout
General Monitor XQuery Functions for Security Administrative Properties Audit

This page allows you to define configuration properties of a data service application.

Access Control

Check Access Control ☐

Allow default anonymous access ☐

Export access control resources

This exports access control resources to a text file.
A third party Security Provider can use this resource information.

Data Cache

Enable Data Cache ✓

Please check the box to enable the data cache in Evaluation Application

Data Cache data source name: cgDataSource

Please select data source JNDI name from the list.

Data Cache table name: MYLDCACHE

Please enter the fully qualified table name. (Default: TABLE_EVALUATION_CACHE)

Purge Data Cache

Server Resources

Max number of query plans cached: 100

3. Enable caching at the function level, by completing the following steps (you can cache both logical and physical data service functions):

   a. Open the CustomerProfile folder, located in Evaluation\DataServices\CustomerManagement. The list of data service functions page opens.

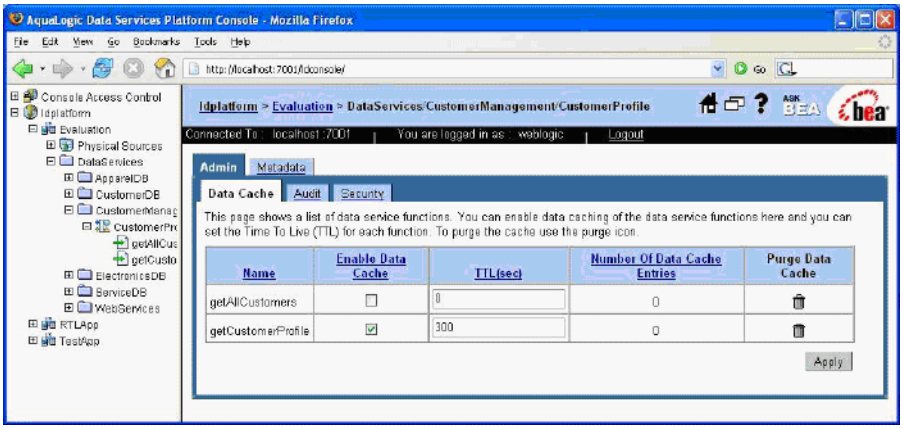
   b. For the `getCustomerProfile()` function, select Enable Cache.

   c. Enter 300 in the TTL (sec) field.

   d. Click Apply.

Note: Application level cache should be enabled.

Figure 15-3 Setting TTL


AquaLogic Data Services Platform Console - Mozilla Firefox
File Edit View Go Bookmarks Tools Help
http://localhost:7001/ldconsole/
Console Access Control
ldplatform
Evaluation
Physical Sources
DataServices
ApparelDB
CustomerDB
CustomerManag
CustomerPr
getAllCus
getCusto
ElectronicsDB
ServiceDB
WebServices
RTLApp
TestApp
ldplatform > Evaluation > DataServices/CustomerManagement/CustomerProfile
Connected To : localhost :7001
You are logged in as : weblogic
Logout
Admin
Metadata
Data Cache
Audit
Security
Apply

This page shows a list of data service functions. You can enable data caching of the data service functions here and you can set the Time To Live (TTL) for each function. To purge the cache use the purge icon.

Name	Enable Data Cache	TTL(sec)	Number Of Data Cache Entries	Purge Data Cache
getAllCustomers	☐	0	0	🗑
getCustomerProfile	✓	300	0	🗑

15.3 Testing the Caching Policy

Testing the caching policy helps you determine whether the specified query results are being cached.

Objectives

In this exercise, you will:

- Use WebLogic Workshop to test the caching policy for the `getCustomerProfile()` function.
- Use the ALDSP Console to verify that the cache is populated.

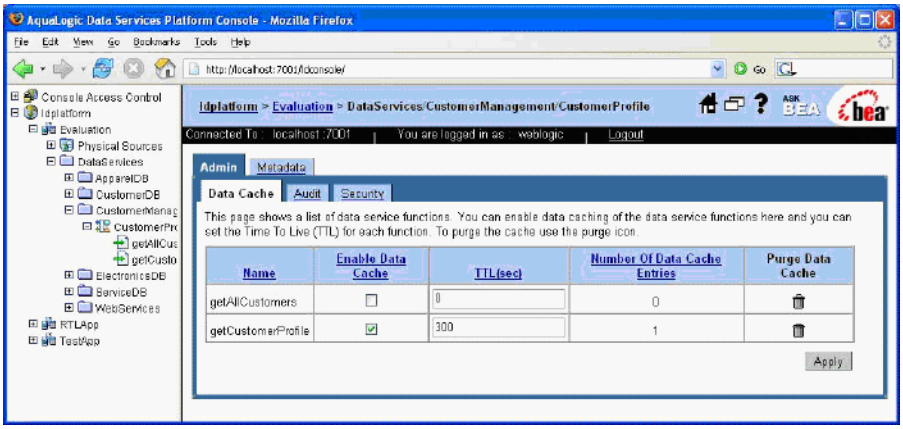
Instructions

1. In WebLogic Workshop, open the CustomerProfile data service in Test View.
2. Select `getCustomerProfile(CustomerID)` from the Function drop-down list.
3. Enter CUSTOMER3 in the Parameter field.
4. Click Execute.
5. In the ALDSP Console, verify that the cache is populated by completing the following steps:

a. Go to the CustomerProfile folder.

b. Confirm that there are entries in the Number of Cache Entries field for the `getCustomerProfile()` function.

Figure 15-4 Cache Test Results in the Metadata Browser


AquaLogic Data Services Platform Console - Mozilla Firefox
File Edit View Go Bookmarks Tools Help
http://localhost:7001/ldconsole/
Console Access Control
ldplatform
Evaluation
Physical Sources
DataServices
ApparelDB
CustomerDB
CustomerManag
CustomerPro
getAllCus
getCusto
ElectronicsDB
ServiceDB
WebServices
RTLApp
TestApp
ldplatform > Evaluation > DataServices/CustomerManagement/CustomerProfile
Connected To : localhost :7001 | You are logged in as : weblogic | Logout
Admin
Metadata
Data Cache
Audit
Security
Apply

This page shows a list of data service functions. You can enable data caching of the data service functions here and you can set the Time To Live (TTL) for each function. To purge the cache use the purge icon.

Name	Enable Data Cache	TTL(sec)	Number Of Data Cache Entries	Purge Data Cache
getAllCustomers	☐	0	0	🗑
getCustomerProfile	✓	300	1	🗑

15.4 Determining Performance Impact of the Caching Policy

A caching policy can reduces processing time and enhance overall system performance.

Objectives

In this exercise, you will:

- Use the PointBase Console to confirm that the cache was populated.
- Use WebLogic Workshop to determine caching performance.

Instructions

1. Use the PointBase Console to verify that the cache was populated, by completing the following steps:

   a. Start the PointBase Console, by entering the following command at the command prompt:

```
$BEA_HOME\weblogic81\common\bin\startPointBaseConsole.cmd
```

b. Enter the following configuration parameters to connect to your local PointBase Console:

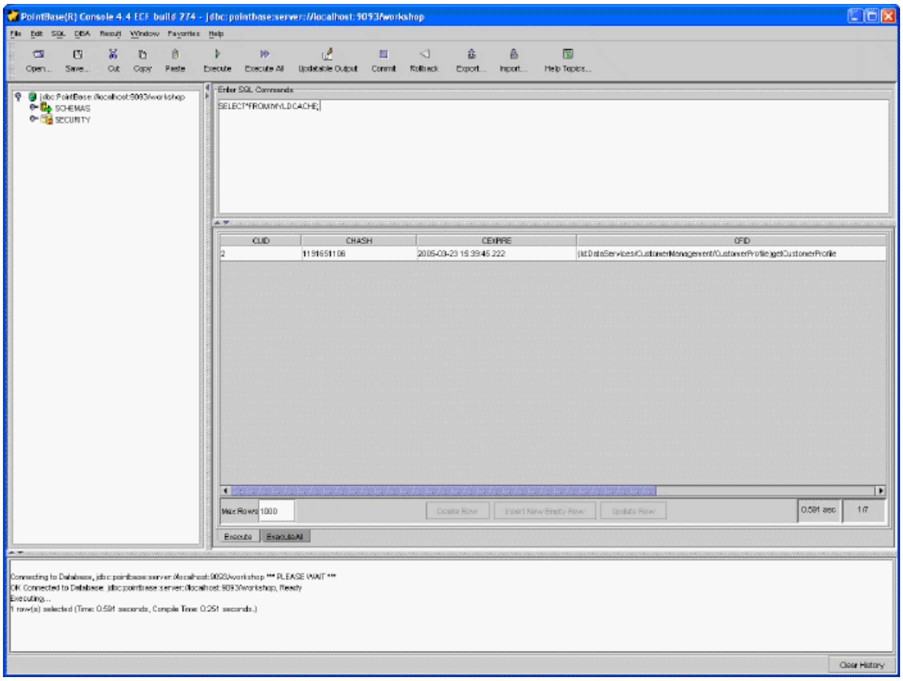
- Driver: com.pointbase.jdbc.jdbcUniversalDriver
- URL: jdbc:pointbase:server://localhost:9093/workshop
- User: weblogic
- Password: weblogic

c. Click OK.

d. Enter the SQL command SELECT * FROM MYLDCACHE to check whether the cache is populated.

e. Click Execute.

Figure 15-5 PointBase Console



```
SELECT*FROM MYLDCACHE;
```

CLID	CHASH	CEXPIRE	CFID
2	1191651106	2005-03-23 15:39:45.222	{ld:DataServices/CustomerManagement/CustomerProfile}getCustomerProfile

```
Connecting to Database, jdbc:pointbase:server://localhost:9093/workshop *** PLEASE WAIT ***
OK Connected to Database: jdbc:pointbase:server://localhost:9093/workshop, Ready
Executing...
1 row(s) selected (Time: 0.591 seconds, Compile Time: 0.251 seconds.)
```

2. In WebLogic Workshop, open the CustomerProfile data service in Test View.

3. Select getCustomerProfile(CustomerID) from the Function drop-down menu.
4. Enter CUSTOMER3 in the Parameter field.
5. Click Execute. The Output window displays the cache's execution time.
6. Use the Output window to determine whether caching helped reduce the query execution time.

15.5 Disable Caching

Caution: For the purposes of the following lessons, you must disable the cache to avoid problems with data updates.

Objectives

In this exercise, you will:

- Disable caching at the application.
- Disable caching at the function level.

Instructions

1. In the ALDSP Console using the + icon, expand the ldplatform directory. (Note: If you click the ldplatform name, the Application List page opens. You do not want this page for this exercise.)
2. Disable application-level caching, by completing the following steps:
   a. Click Evaluation. The ALDSP Console's General page opens.
   b. In the Data Cache section, clear Enable Data Cache.
   c. Click Apply.
3. Disable function-level caching, by completing the following steps:
   a. Open the CustomerProfile folder, located in

   `Evaluation\DataServices\CustomerManagement`

   The list of data service functions page opens.
   b. For the `getCustomerProfile()` function, clear Enable Data Cache.
   c. Click Apply.

Lesson Summary

In this lesson, you learned how to:

- Use the ALDSP Console to configure the ALDSP cache.
- Enable the cache for a data service function and define its time-to-live (TTL).
- Check the database to verify whether a cache is used.
- Determine the performance impact of the cache, by checking the query response time.
- Disable caching.

Managing Data Service Security

The Data Services Platform (ALDSP) leverages the security features of the underlying WebLogic platform. Specifically, it uses resource authorization to control access to ALDSP resources based on user identity or other information.

Note: WebLogic Server must be running.

Objectives

After completing this lesson, you will be able to:

- Enable application-level security.
- Set function-level read and write access security.
- Set element-level security.

Overview

ALDSP's security infrastructure extends WebLogic Server's security policies to include ALDSP objects such as data sources and stored queries, as well as security roles, groups, and users. These security policies allow ALDSP administrators to set up rules that dynamically determine whether a given user:

- Can access a particular object.
- Holds read/write/execute permissions on a ALDSP object or a subset of those permissions.

By default data services do not have any security policies configured. Therefore data is generally accessible unless a more restrictive policy for the information is configured. Security policies can apply at various levels of granularity, including:

- **Application level.** The policy applies to all data services within the deployed ALDSP application.
- **Data service level.** The policy applies to individual data services within the application.
- **Element level.** A policy can apply to individual items of information within a return type, such as a salary node in a customer object. If blocked by insufficient credentials at this level, the rest of the returned information is provided without the blocked element.

Implementing ALDSP access control involves using the WebLogic Server Console to configure user groups and roles. You can then use the ALDSP Console to create policies for ALDSP, including data services and their functions.

16.1 Creating New User Accounts

The first step in creating data service security policies is to create user accounts and either assign the user account to a default group or configure a new group. There are 12 default authenticator groups.

Objectives

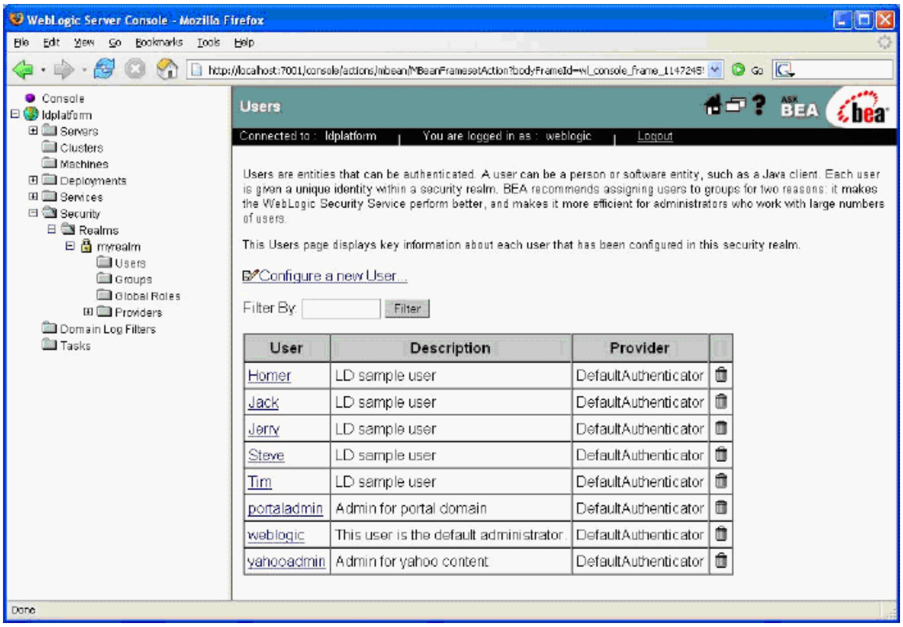
In this exercise, you will:

- Open the WebLogic Server Console.
- Create two user accounts that use a default user group.
- View the user list.

Instructions

1. Open the WebLogic Server Console (`http://localhost:7001/console/`), using the following credentials:
    - User Name = weblogic
    - Password = weblogic
2. Choose Security → Realms → myrealm → Users.

Figure 16-1 User Security


WebLogic Server Console - Mozilla Firefox
File Edit View Go Bookmarks Tools Help
http://localhost:7001/console/actions/mbean/MBeanFramesetAction?bodyFrameId=wl_console_frame_1147245
Console
ldplatform
Servers
Clusters
Machines
Deployments
Services
Security
Realms
myrealm
Users
Groups
Global Roles
Providers
Domain Log Filters
Tasks
Users
Connected to : ldplatform
You are logged in as : weblogic
Logout
Configure a new User...
Filter By:
Filter
Done

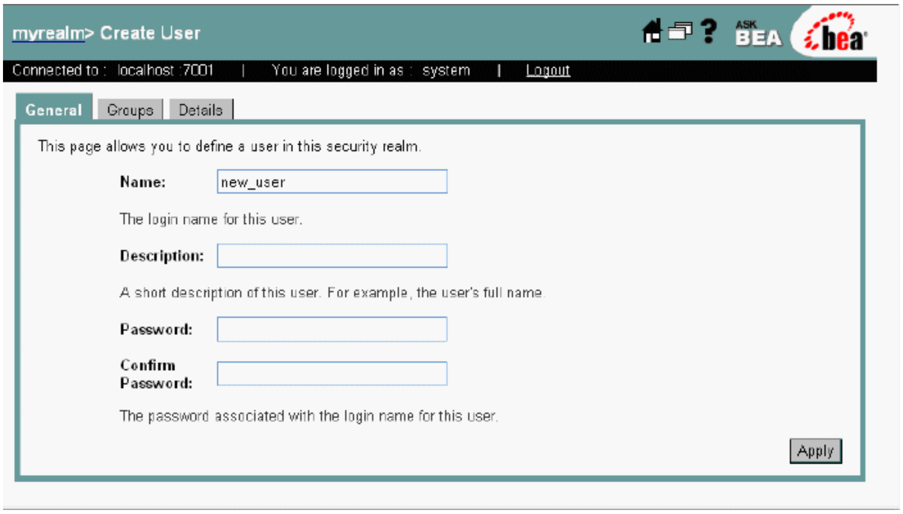
Users are entities that can be authenticated. A user can be a person or software entity, such as a Java client. Each user is given a unique identity within a security realm. BEA recommends assigning users to groups for two reasons: it makes the WebLogic Security Service perform better, and makes it more efficient for administrators who work with large numbers of users.

This Users page displays key information about each user that has been configured in this security realm.

User	Description	Provider	
Homer	LD sample user	DefaultAuthenticator	🗑
Jack	LD sample user	DefaultAuthenticator	🗑
Jerry	LD sample user	DefaultAuthenticator	🗑
Steve	LD sample user	DefaultAuthenticator	🗑
Tim	LD sample user	DefaultAuthenticator	🗑
portaladmin	Admin for portal domain	DefaultAuthenticator	🗑
weblogic	This user is the default administrator.	DefaultAuthenticator	🗑
yahooadmin	Admin for yahoo content	DefaultAuthenticator	🗑

3. Select Configure New User.

Figure 16-2 Define User in Security Realm


myrealm> Create User
ASK BEA
bea
Connected to : localhost :7001 | You are logged in as : system | Logout
General
Groups
Details
Apply

This page allows you to define a user in this security realm.

Name: new_user

The login name for this user.

Description:

A short description of this user. For example, the user's full name.

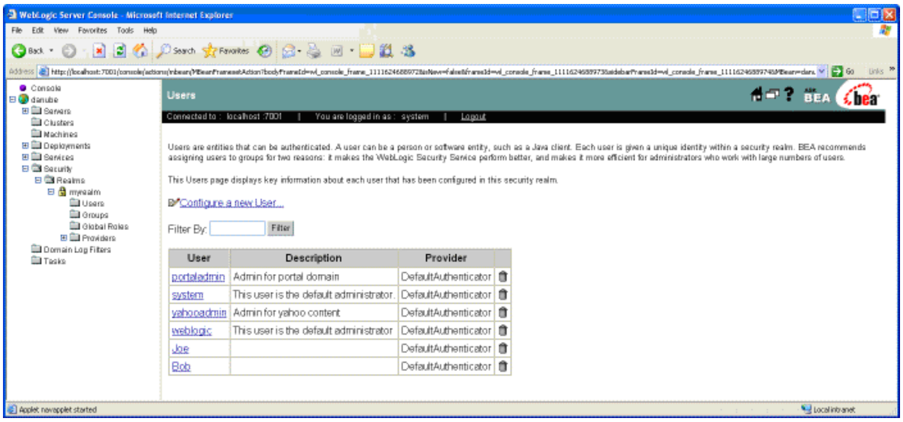
Password:

Confirm Password:

The password associated with the login name for this user.

4. Create a new user account by completing the following steps:
   a. Enter Joe in the Name field.
   b. Enter password in the Password field.
   c. Enter password in the Confirm Password field.
   d. Click Apply.
5. Repeat step 3 and step 4, entering Bob in the Name field (step 4a).
6. (*Optional*) Choose Security → Realms → myrealm → Users to view the results.

Figure 16-3 New Users Added



Connected to : localhost :7001 | You are logged in as : system

Users are entities that can be authenticated. A user can be a person or software entity, such as a Java client. Each user is given a unique identity within a security realm. BEA recommends assigning users to groups for two reasons: it makes the WebLogic Security Service perform better, and makes it more efficient for administrators who work with large numbers of users.

This Users page displays key information about each user that has been configured in this security realm.

User	Description	Provider	
portaladmin	Admin for portal domain	DefaultAuthenticator	🗑
system	This user is the default administrator.	DefaultAuthenticator	🗑
yahooadmin	Admin for yahoo content	DefaultAuthenticator	🗑
weblogic	This user is the default administrator	DefaultAuthenticator	🗑
Joe		DefaultAuthenticator	🗑
Bob		DefaultAuthenticator	🗑

16.2 Setting Application-Level Security

Application-level security applies to all data services within the deployed ALDSP domain, regardless of user permission or group. By default, when you turn on access control for an application, access to any of its resources is blocked, except for users who comply with policies configured for the resources.

Alternatively, by allowing default anonymous access, you can grant access to its resources by default. You can enable default anonymous access level by navigating to Application level General tab under Access Control (application Name → General). In this case, a resource is restricted only if a more specific security policy for it exists; for example, a security policy at the data service function level.

Objectives

In this exercise, you will:

- Use the AquaLogic Data Services Platform Console to enable application-level security.
- Use WebLogic Workshop to test the security policy.

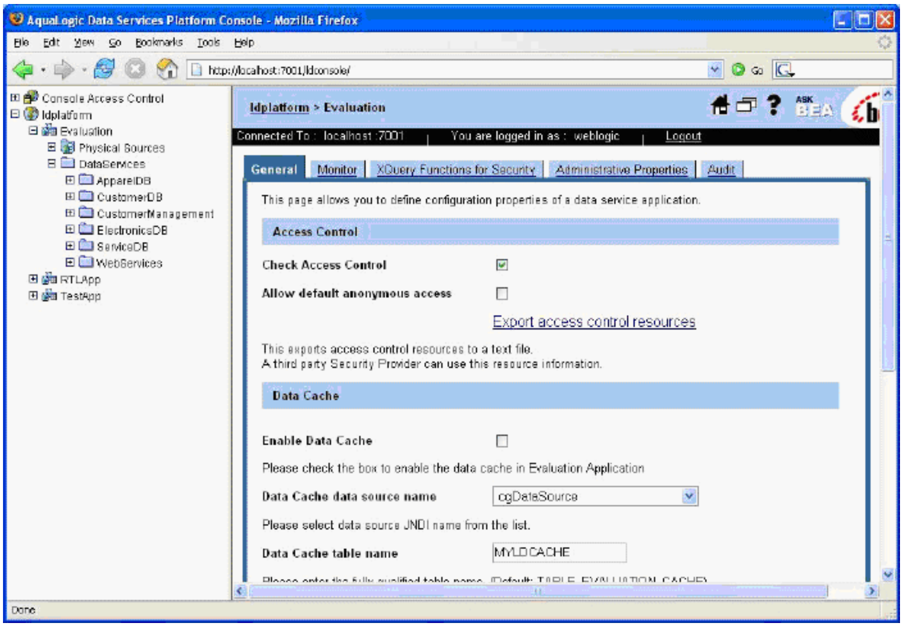
Instructions

1. In the ALDSP Console (`http://localhost:7001/ldconsole/`), using the + icon, expand the ldplatform directory.

Note: If you click the ldplatform name, the Application List page opens. You do not want this page for this lesson.

2. Click Evaluation. The application's General page opens.
3. Select Check Access Control.
4. Click Apply.

Figure 16-4 Set General Security


AquaLogic Data Services Platform Console - Mozilla Firefox
File Edit View Go Bookmarks Tools Help
http://localhost:7001/ldconsole/
Console Access Control
ldplatform
Evaluation
Physical Sources
DataServices
ApparelDB
CustomerDB
CustomerManagement
ElectronicsDB
ServiceDB
WebServices
RTLApp
TestApp
ldplatform > Evaluation
Connected To : localhost :7001
You are logged in as : weblogic
Logout
General
Monitor
XQuery Functions for Security
Administrative Properties
Audit
Export access control resources
Done

This page allows you to define configuration properties of a data service application.

Access Control

Check Access Control ✓

Allow default anonymous access ☐

This exports access control resources to a text file.
A third party Security Provider can use this resource information.

Data Cache

Enable Data Cache ☐

Please check the box to enable the data cache in Evaluation Application

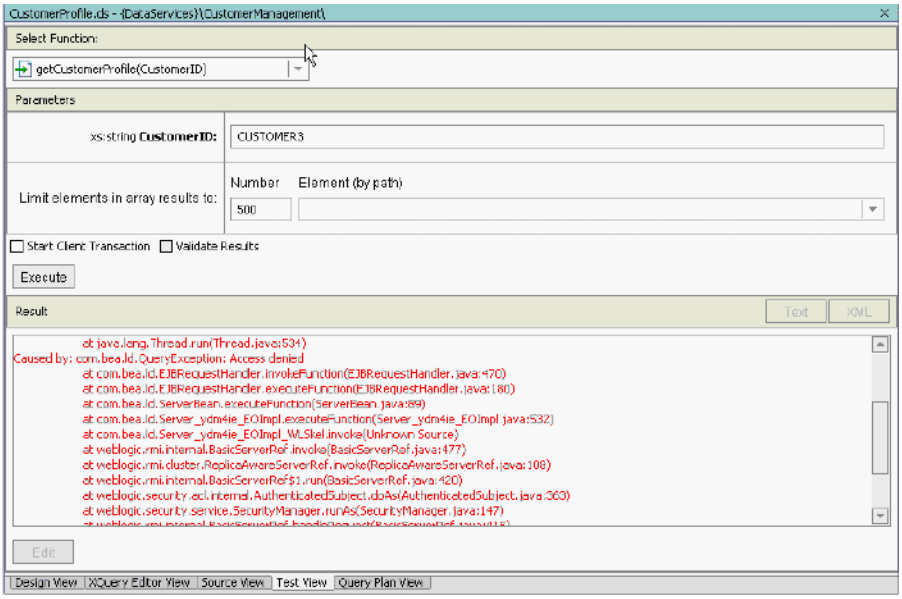
Data Cache data source name: cgDataSource

Please select data source JNDI name from the list.

Data Cache table name: MYLDCACHE

5. Test the security policy by completing the following steps:
   a. In WebLogic Workshop, open `CustomerProfile.ds` in Test View.
   b. Select `getCustomerProfile()` from the Function drop-down list.
   c. Enter CUSTOMER3 in the Parameters field.
   d. Click Execute. The test should return an Access Denied error. With the current security settings, no one can access the application's functions. You must grant user access to read and write functions.

Figure 16-5 Access Denied



xs:string **CustomerID:** CUSTOMER3

Number: 500

```
                at java.lang.Thread.run(Thread.java:534)
Caused by: com.bea.ld.QueryException: Access denied
                at com.bea.ld.EJBRequestHandler.invokeFunction(EJBRequestHandler.java:470)
                at com.bea.ld.EJBRequestHandler.executeFunction(EJBRequestHandler.java:180)
                at com.bea.ld.ServerBean.executeFunction(ServerBean.java:89)
                at com.bea.ld.Server_ydm4ie_EOImpl.executeFunction(Server_ydm4ie_EOImpl.java:532)
                at com.bea.ld.Server_ydm4ie_EOImpl_WLSkel.invoke(Unknown Source)
                at weblogic.rmi.internal.BasicServerRef.invoke(BasicServerRef.java:477)
                at weblogic.rmi.cluster.ReplicaAwareServerRef.invoke(ReplicaAwareServerRef.java:108)
                at weblogic.rmi.internal.BasicServerRef$1.run(BasicServerRef.java:420)
                at weblogic.security.acl.internal.AuthenticatedSubject.doAs(AuthenticatedSubject.java:363)
                at weblogic.security.service.SecurityManager.runAs(SecurityManager.java:147)
```

16.3 Granting User Access to Read Functions

ALDSP security policies can be set at the function level, which applies to specific functions within specific data services. Function-level security can be read and/or write permissions. A policy may include any number of restrictions; for example, limiting access based on the role of the user or on the time of access. Specifically, policies can be based on the following criteria:

- **User Name of the Caller.** Creates a condition for a security policy based on a user name. For example, you might create a condition indicating that only the user John can access the Customer data service.
- **Caller is a Member of the Group.** Creates a condition for a security policy based on a group.
- **Caller is Granted the Role.** Creates a condition based on a security role. A security role is a special type of user group specifically for applying and managing common security needs of a group of users.
- **Hours of Access are Between.** Creates a condition for a security policy based on a specified time period.

- **Server is in Development Mode.** Creates a condition for a security policy based on whether the server is running in development mode.

Objectives

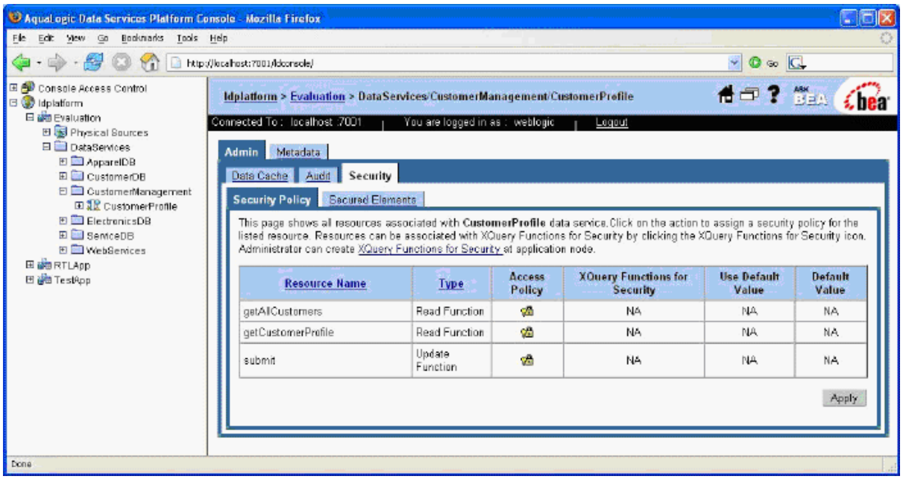
In this exercise, you will:

- Use the ALDSP Console to grant Joe read access permissions, based on user name.
- Use WebLogic Workshop to test the new security policy.

Instructions

1. In the ALDSP Console, open the CustomerProfile data service. (The data service is located in ldplatform\Evaluation\DataServices\CustomerManagement.)
2. Click the Security tab. The Security Policy tab opens.

Figure 16-6 Data Service-Level Security Policy

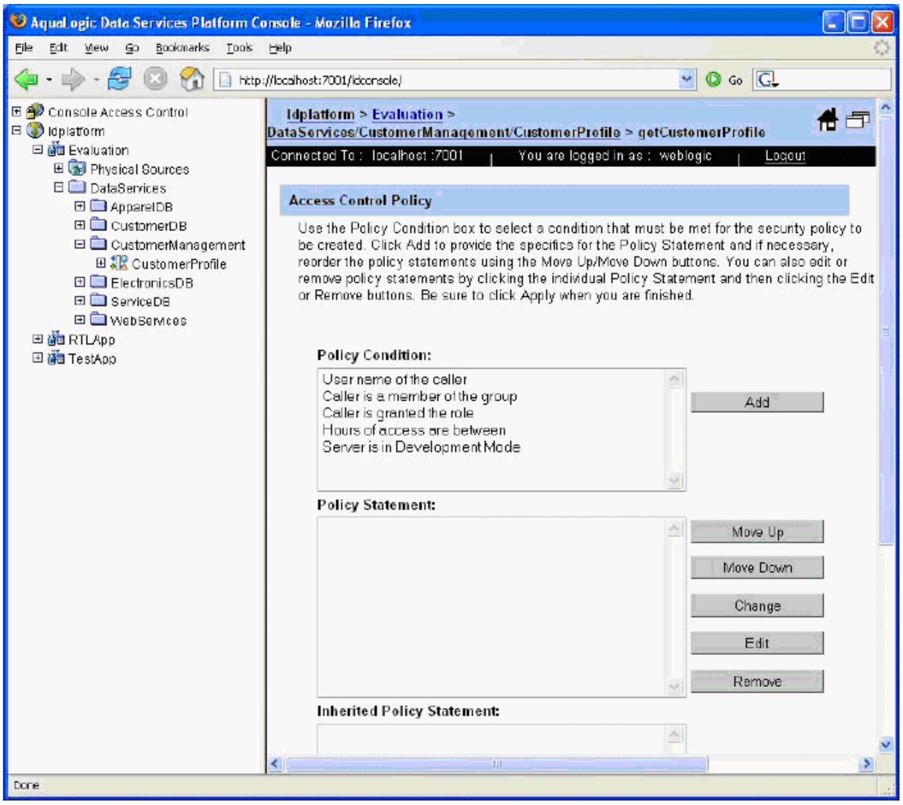

AquaLogic Data Services Platform Console - Mozilla Firefox
File Edit View Go Bookmarks Tools Help
http://localhost:7001/ldconsole/
Console Access Control
ldplatform
Evaluation
Physical Sources
DataServices
ApparelDB
CustomerDB
CustomerManagement
CustomerProfile
ElectronicsDB
ServiceDB
WebServices
RTLApp
TestApp
ldplatform > Evaluation > DataServices/CustomerManagement/CustomerProfile
Connected To : localhost :7001 | You are logged in as : weblogic | Logout
Admin
Metadata
Data Cache
Audit
Security
Security Policy
Secured Elements
Apply
Done

This page shows all resources associated with **CustomerProfile** data service. Click on the action to assign a security policy for the listed resource. Resources can be associated with XQuery Functions for Security by clicking the XQuery Functions for Security icon. Administrator can create XQuery Functions for Security at application node.

Resource Name	Type	Access Policy	XQuery Functions for Security	Use Default Value	Default Value
getAllCustomers	Read Function	🔒	NA	NA	NA
getCustomerProfile	Read Function	🔒	NA	NA	NA
submit	Update Function	🔒	NA	NA	NA

3. Click the Action Policy icon for the getCustomerProfile resource to open the Access Control Policy window.

Figure 16-7 Configure Security

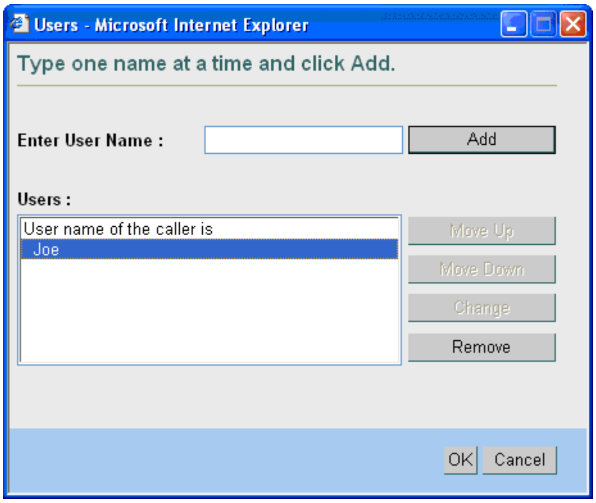

AquaLogic Data Services Platform Console - Mozilla Firefox
File Edit View Go Bookmarks Tools Help
http://localhost:7001/ldconsole/
Console Access Control
ldplatform
Evaluation
Physical Sources
DataServices
ApparelDB
CustomerDB
CustomerManagement
CustomerProfile
ElectronicsDB
ServiceDB
WebServices
RTLApp
TestApp
ldplatform > Evaluation > DataServices/CustomerManagement/CustomerProfile > getCustomerProfile
Connected To : localhost :7001 | You are logged in as : weblogic | Logout
Access Control Policy
Policy Condition:
Add
Policy Statement:
Move Up
Move Down
Change
Edit
Remove
Inherited Policy Statement:
Done

Use the Policy Condition box to select a condition that must be met for the security policy to be created. Click Add to provide the specifics for the Policy Statement and if necessary, reorder the policy statements using the Move Up/Move Down buttons. You can also edit or remove policy statements by clicking the individual Policy Statement and then clicking the Edit or Remove buttons. Be sure to click Apply when you are finished.

User name of the caller
Caller is a member of the group
Caller is granted the role
Hours of access are between
Server is in Development Mode

4. Set read access for a specific user, by completing the following steps:
   a. Select User name of the caller.
   b. Click Add. The Users dialog box opens.
   c. Enter Joe in the Name field.
   d. Click Add.

Figure 16-8 Adding User


Users - Microsoft Internet Explorer
Type one name at a time and click Add.
Enter User Name :
Add
Users :
User name of the caller is
Joe
Move Up
Move Down
Change
Remove
OK
Cancel

   e. Click OK and move back to the Access Control Policy window.

   f. Click Apply.

5. Login to the now-secure application, by completing the following steps:

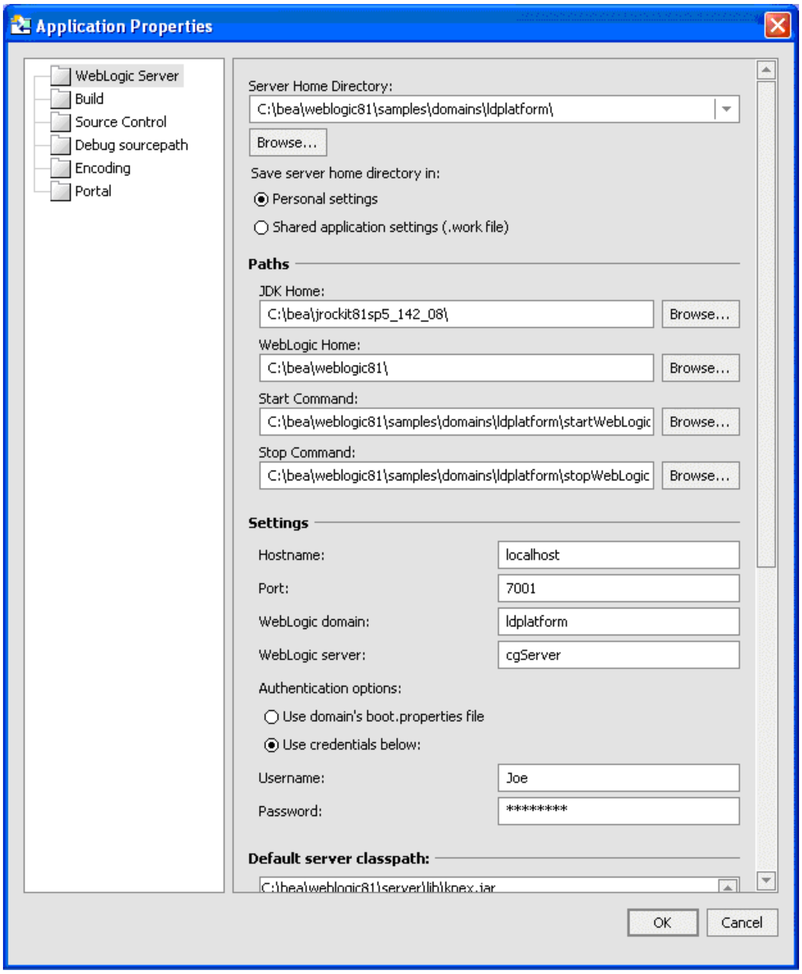
   a. In WebLogic Workshop, choose Tools → Application Properties → WebLogic Server.

   b. Select Use Credentials Below.

   c. Enter Joe and password in the Use Credentials Below fields.

   d. Click OK.

Figure 16-9 Logging Into Secure Application



Server Home Directory: C:\bea\weblogic81\samples\domains\ldplatform\

Save server home directory in: [✓ Personal settings] Shared application settings (.work file)

Paths

JDK Home: C:\bea\jrockit81sp5_142_08\

WebLogic Home: C:\bea\weblogic81\

Start Command: C:\bea\weblogic81\samples\domains\ldplatform\startWebLogic

Stop Command: C:\bea\weblogic81\samples\domains\ldplatform\stopWebLogic

Settings

Hostname:	localhost
Port:	7001
WebLogic domain:	ldplatform
WebLogic server:	cgServer

Authentication options: Use domain's boot.properties file [✓ Use credentials below:]

Username:	Joe
Password:	********

Default server classpath:

C:\bea\weblogic81\server\lib\knex.jar

6. Test the security policy by completing the following steps:

   a. Open `CustomerProfile.ds` in Test View.

   b. Select `getCustomerProfile()` from the Function drop-down list.

   c. Enter CUSTOMER3 in the Parameters field.

   d. Click Execute. The test should permit access and return the requested data.

e. Click Edit, modify an item, and then click Submit. An error message will display because Joe is granted only read access.

16.4 Granting User Access to Write Functions

As noted in the previous exercise, security policies at the function level can allow either read and/or write permissions.

Objectives

In this exercise, you will:

- Use the ALDSP Console to grant Joe write access permissions.
- Use WebLogic Workshop to test the new security policy.

Instructions

1. In the ALDSP Console, open the CustomerProfile data service.
2. Select the Security tab. The Security Policy tab opens.
3. Click the Action Policy icon for the submit resource. The Access Control Policy window opens.
4. Set write access to a user, by completing the following steps:
   a. Select User name of the caller.
   b. Click Add.
   c. Enter Joe in the Name field.
   d. Click Add.
   e. Click OK.
   f. Click Apply.
5. Test the security policy, by completing the following steps:
   a. In WebLogic Workshop, open `CustomerProfile.ds` in Test View. The file is located in DataServices\CustomerManagement.
   b. Select `getCustomerProfile()` from the Function drop-down list.

c. Enter CUSTOMER3 in the Parameters field.

d. Click Execute. The test should permit access and return the specified results.

e. Click Edit. Because Joe is granted both read and write access, you can now edit the results.

16.5 Setting Element-Level Data Security

A policy can apply to individual items of information within a return type, such as a salary node in a customer object. If blocked by insufficient credentials at this level, the rest of the returned information is provided without the blocked element.

Objectives

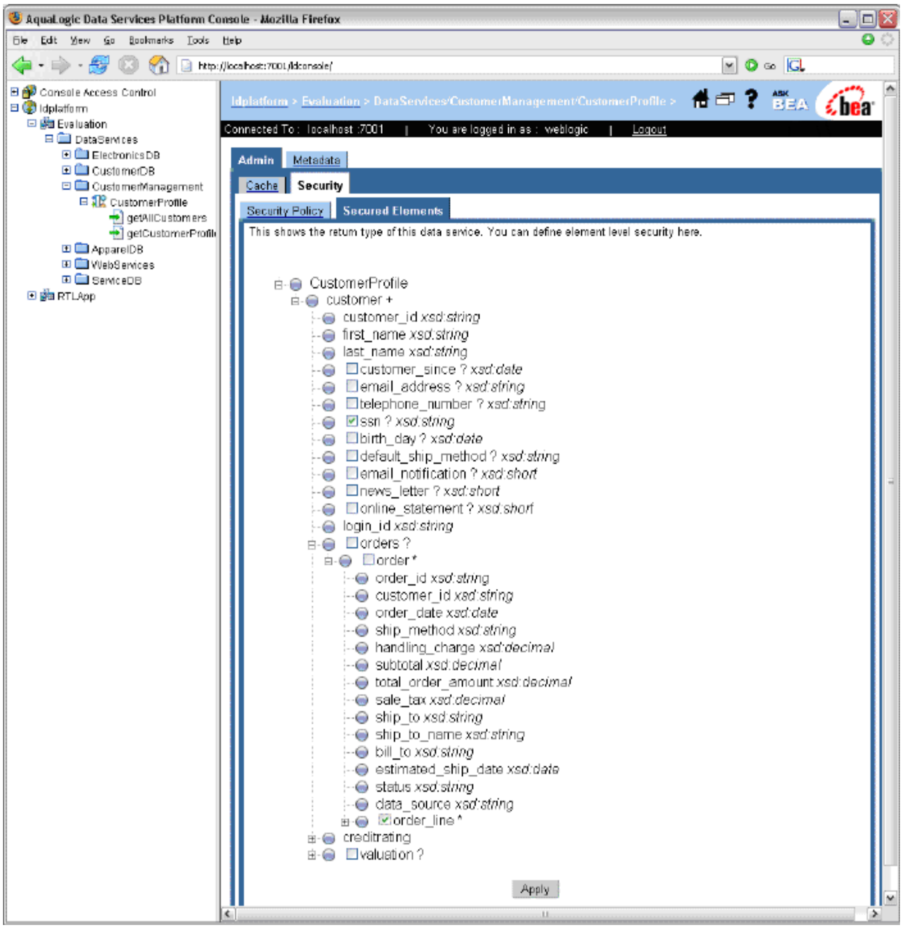
In this exercise, you will:

- Enable element-level security.
- Set a security policy for specific elements.

Instructions

1. In the ALDSP Console, open the CustomerProfile data service.
2. Select the Security tab.
3. Set element-level security, by completing the following steps:
   a. Select the Secured Elements tab.
   b. Expand the CustomerProfile and customer+ nodes.
   c. Select the checkbox for the ssn simple element.
   d. Expand the orders ? and orders * nodes.
   e. Select the checkbox for the order_line * complex element.
   f. Click Apply.

Figure 16-10 Setting Element-Level Security

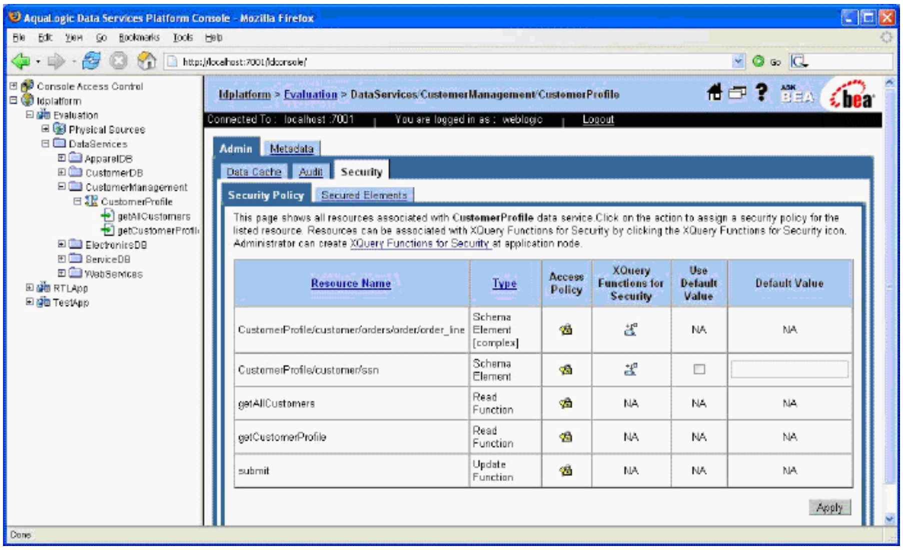

AquaLogic Data Services Platform Console - Mozilla Firefox
File Edit View Go Bookmarks Tools Help
http://localhost:7001/ldconsole/
Console Access Control
ldplatform
Evaluation
DataServices
ElectronicsDB
CustomerDB
CustomerManagement
CustomerProfile
getAllCustomers
getCustomerProfile
ApparelDB
WebServices
ServiceDB
RTLApp
ldplatform > Evaluation > DataServices/CustomerManagement/CustomerProfile >
Connected To : localhost :7001 | You are logged in as : weblogic | Logout
Admin
Metadata
Cache
Security
Security Policy
Secured Elements
Apply

This shows the return type of this data service. You can define element level security here.

```
CustomerProfile
  customer +
    customer_id xsd:string
    first_name xsd:string
    last_name xsd:string
    ☐ customer_since ? xsd:date
    ☐ email_address ? xsd:string
    ☐ telephone_number ? xsd:string
    ☑ ssn ? xsd:string
    ☐ birth_day ? xsd:date
    ☐ default_ship_method ? xsd:string
    ☐ email_notification ? xsd:short
    ☐ news_letter ? xsd:short
    ☐ online_statement ? xsd:short
    login_id xsd:string
    ☐ orders ?
      ☐ order *
        order_id xsd:string
        customer_id xsd:string
        order_date xsd:date
        ship_method xsd:string
        handling_charge xsd:decimal
        subtotal xsd:decimal
        total_order_amount xsd:decimal
        sale_tax xsd:decimal
        ship_to xsd:string
        ship_to_name xsd:string
        bill_to xsd:string
        estimated_ship_date xsd:date
        status xsd:string
        data_source xsd:string
        ☑ order_line *
    creditrating
    ☐ valuation ?
```

4. Return to the Security Policy tab for CustomerProfile. You should see two new resources: CustomerProfile/customer/ssn and CustomerProfile/customer/orders/order/order_line.

Figure 16-11 New Secured Element Resources



This page shows all resources associated with **CustomerProfile** data service.Click on the action to assign a security policy for the listed resource. Resources can be associated with XQuery Functions for Security by clicking the XQuery Functions for Security icon. Administrator can create XQuery Functions for Security at application node.

Resource Name	Type	Access Policy	XQuery Functions for Security	Use Default Value	Default Value
CustomerProfile/customer/orders/order/order_line	Schema Element [complex]	🔒	⚙	NA	NA
CustomerProfile/customer/ssn	Schema Element	🔒	⚙	☐	
getAllCustomers	Read Function	🔒	NA	NA	NA
getCustomerProfile	Read Function	🔒	NA	NA	NA
submit	Update Function	🔒	NA	NA	NA

5. Set the security policy for the complex order_line element, by completing the following steps:
   a. Return to the Security Policy tab for CustomerProfile.
   b. Click the Action Policy icon for the CustomerProfile/customer/orders/order/order_line resource. The Access Control Policy window opens.
   c. Select User name of the caller.
   d. Click Add.
   e. Enter Joe in the Name field.
   f. Click Add.
   g. Click OK.
   h. Click Apply.
6. Set the security policy for the simple ssn element, by completing the following steps:
   a. Click the Action Policy icon for the CustomerProfile/customer/ssn resource. The Access Control Policy window opens.

b. Select User name of the caller.
c. Click Add.
d. Enter Bob in the Name field.
e. Click Add.
f. Click OK.
g. Click Apply.

16.6 Testing Element-Level Security

At this point, element-level security policies are defined for both Bob and Joe. Testing the policy within WebLogic Workshop lets you determine what data results these two users will be able to access.

Objectives

In this exercise, you will:

- Test the security policy for Bob and Joe.
- Change the security policy for Bob and test the new security policy.

Instructions

1. Test element-level security for Joe, by completing the following steps:
   a. In WebLogic Workshop, open `CustomerProfile.ds` in Test View.
   b. Select `getCustomerProfile()` from the Function drop-down list.
   c. Enter CUSTOMER3 in the Parameters field.
   d. Click Execute. The test should permit access and return all results except SSN.
   e. Click Edit, modify an order_line value, click Submit, and click OK. The specified change is submitted.
   f. Click Execute to refresh the data set.
   g. Verify that changes have been saved.
2. Test the element-level security policy for Bob, by completing the following steps:

a. Choose Tools → Application Properties → WebLogic Server.

b. Select Use Credentials Below.

c. Enter Bob and password in the Use Credentials Below fields.

d. Click OK.

e. Open `CustomerProfile.ds` in Test View.

f. Select `getCustomerProfile(CustomerID)` from the Function drop-down list.

g. Enter CUSTOMER3 in the Parameters field.

h. Click Execute. The test should fail. Although Bob can access the SSN element, he does not have read access to the `getCustomerProfile()` function.

3. Change the security policy for Bob, by completing the following steps:

a. In the ALDSP Console, open the CustomerProfile data service.

b. Select the Security tab.

c. Click the Action Policy icon for the getCustomerProfile resource. The Access Control Policy window opens.

d. Set read access for Bob, by completing the following steps:

- Select the caller's User name.Click Add.

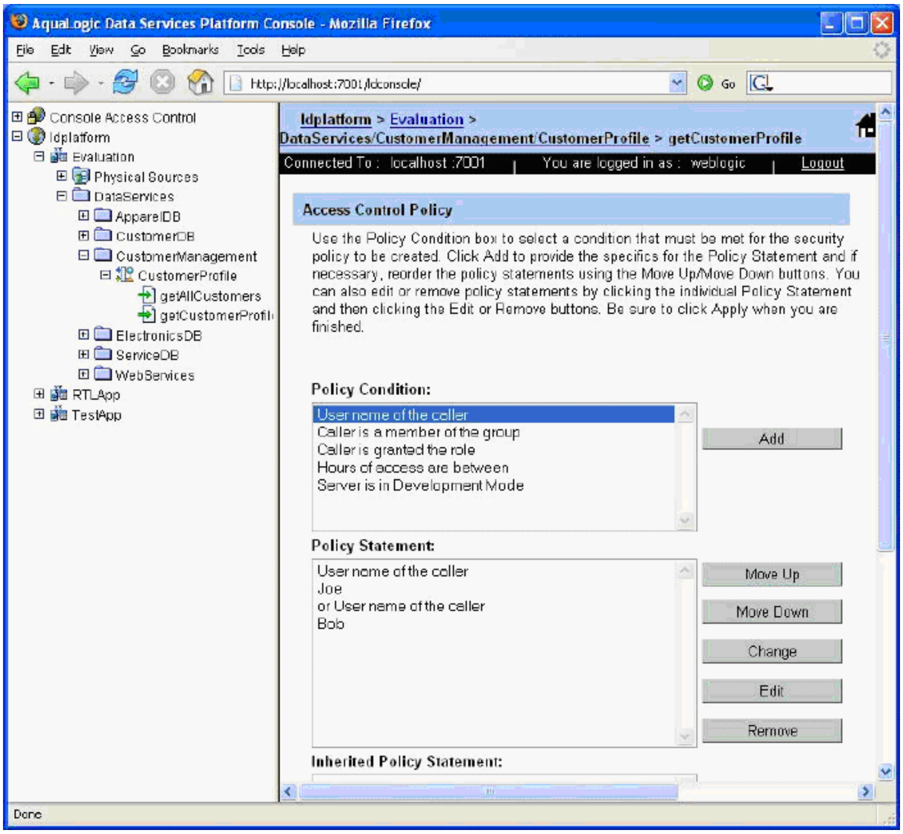
- Enter Bob in the Name field. Click Add, then Ok.

- Click the "and User name of the caller" line, located in the Policy Statement section of the window.

- Click Change, which changes the line to an "or User name of the caller" condition.

- Click Apply.

Figure 16-12 Enabling read Access for Two Users


AquaLogic Data Services Platform Console - Mozilla Firefox

http://localhost:7001/ldconsole/

ldplatform > Evaluation > DataServices/CustomerManagement/CustomerProfile > getCustomerProfile

Connected To : localhost :7001 | You are logged in as : weblogic | Logout

Access Control Policy

Use the Policy Condition box to select a condition that must be met for the security policy to be created. Click Add to provide the specifics for the Policy Statement and if necessary, reorder the policy statements using the Move Up/Move Down buttons. You can also edit or remove policy statements by clicking the individual Policy Statement and then clicking the Edit or Remove buttons. Be sure to click Apply when you are finished.

Policy Condition:

User name of the caller
Caller is a member of the group
Caller is granted the role
Hours of access are between
Server is in Development Mode

Policy Statement:

User name of the caller
Joe
or User name of the caller
Bob

Inherited Policy Statement:

4. In WebLogic Workshop, test the `getCustomerProfile()` function again. This time, user Bob can view all elements except order_line information.

5. Try modifying data by clicking on Edit button and changing SSN. Submit changes by clicking on Submit button. An error message will display because Bob does not have write privileges.

6. Reset the application-level security, by completing the following steps:

   a. Reset the WebLogic Workshop → Tools → Application Properties → WebLogic Server authentication options back to user: weblogic, password: weblogic.

   b. In the ALDSP Console (`http://localhost:7001/ldconsole/`), using the + icon, expand the ldplatform directory.

Note: If you click the ldplatform name, the Application List page opens. You do not need this page for this lesson.

c. Click Evaluation. The Administration Control's General page opens.

d. Clear Check Access Control.

e. Click Apply.

Lesson Summary

In this lesson, you learned how to:

- Activate application level security.
- Set security permissions on both read and write function access.
- Set security permissions on simple and complex elements.

(Optional) Consuming Data Services through Portals & Business Processes

The previous lessons demonstrated how ALDSP provides a convenient way to quickly access ALDSP from a WebLogic Workshop application such as page flows, process definitions, or portals. This optional lesson details the steps you take to use a portal to access data services.

Note: WebLogic Portal must be installed.

Objectives

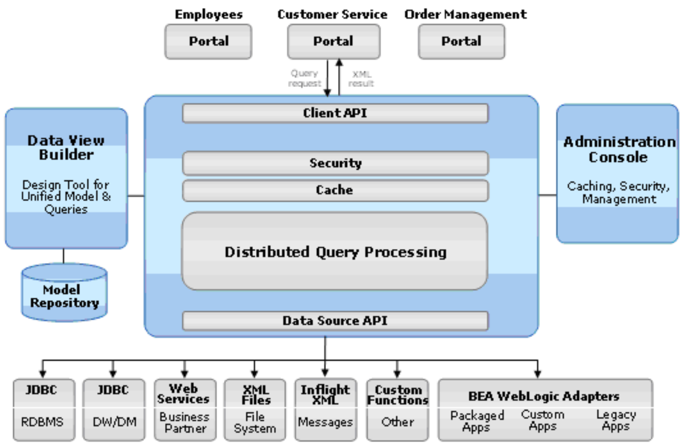
After completing this lesson, you will be able to:

- Import a WebLogic Portal project that contains portals and business processes.
- Install the Data Service Control in the project, thereby making data services available from the portal and business processes.
- Recognize how a Data Service Control is used from a portal and business process.

Overview

At its most basic level, a portal is a Web site that simplifies and personalizes access to content, applications, and processes. Technically speaking, a portal is a container of resources and functionality that can be made available to end-users. These portal views, which are called Desktops in WebLogic Portal, provide the uniform resource location (URL) that end users access.

Figure 17-1 Consuming Data Services from Portals



17.1 Installing a Data Service Control in a Portal Project

The steps within this exercise are similar to those detailed in Installing a Data Service Control.

Objectives

In this exercise, you will:

- Import a portal web project's files and libraries, which you will use to create a new portal project.
- Create a new portal project.
- Add a control to the portal project.

Instructions

1. Right-click the Evaluation application.
2. Choose Install → Portal. ALDSP installs the necessary portal files and libraries.
3. Create a new portal web project by completing the following steps:
   a. Right-click the Evaluation application.

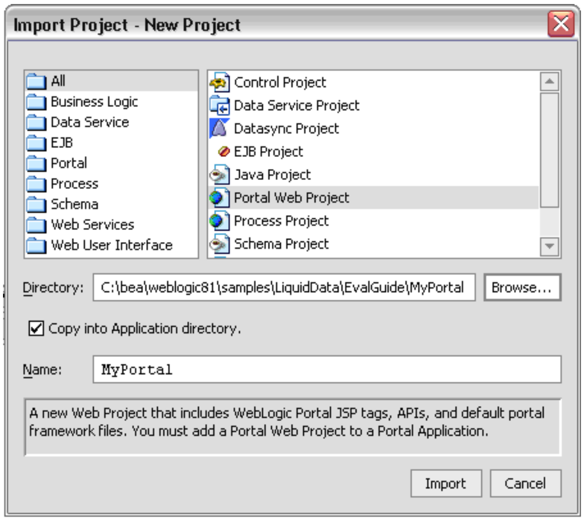
b. Choose Import Project.

c. Select Portal Web Project.

d. Select MyPortal, located in the <beahome>\weblogic81\samples\ liquiddata\EvalGuide directory.

e. Click Open and then click Import.

Figure 17-2 Importing a Portal Web Project



Directory: C:\bea\weblogic81\samples\LiquidData\EvalGuide\MyPortal

☑ Copy into Application directory.

Name: MyPortal

A new Web Project that includes WebLogic Portal JSP tags, APIs, and default portal framework files. You must add a Portal Web Project to a Portal Application.

4. Create a new folder in the MyPortal folder, and name it controls.

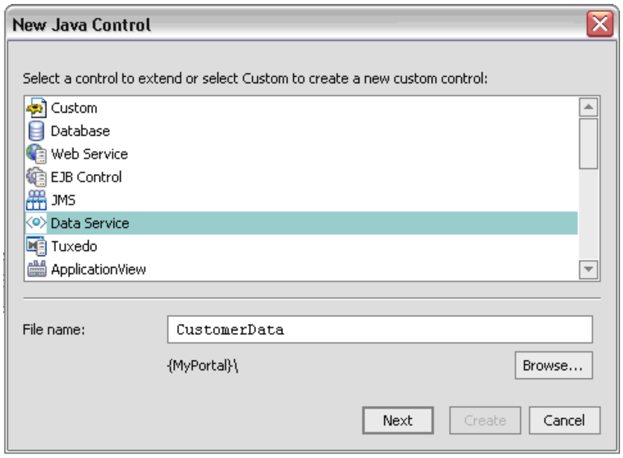
5. Create a Data Service Control within the portal by completing the following steps:

   a. Right-click the MyPortal project.

   b. Choose **New** → **Java Control**.

   c. Select **Data Service Control** and name it CustomerData.

Figure 17-3 Creating a New Data Service Control



Select a control to extend or select Custom to create a new custom control: Data Service

File name: CustomerData

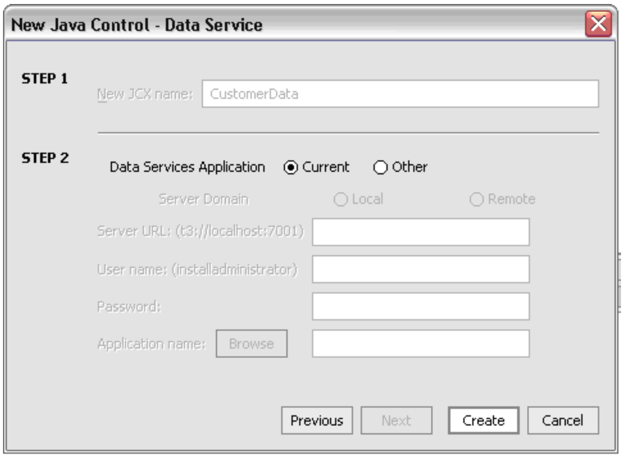
{MyPortal}\

d. Click **Next** and then click **Yes** at the Message window.

e. Select MyPortal\controls as the subfolder in which to locate the new control.

f. Click **Select**. The New Java Control → ALDSP window opens.

Figure 17-4 Setting Data Service Control Specifications


New Java Control - Data Service
STEP 1
STEP 2
Server Domain
Local
Remote
Server URL: (t3://localhost:7001)
User name: (installadministrator)
Password:
Application name:
Browse
Previous
Next
Create
Cancel

New JCX name: CustomerData

Data Services Application [✓ Current] Other

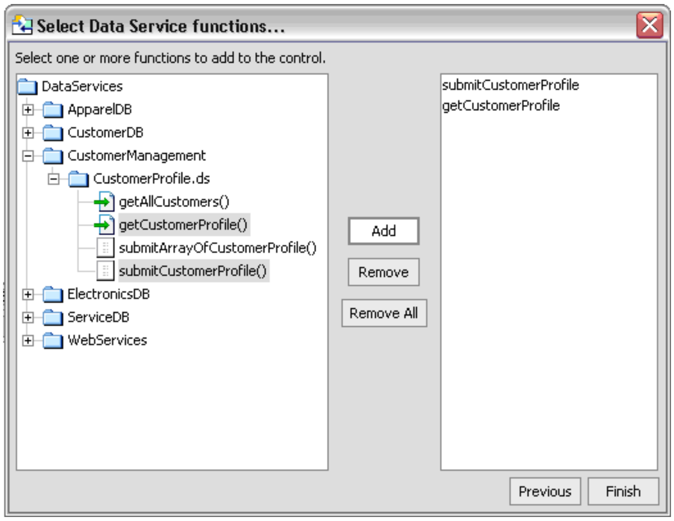
g. Click Create to accept the default settings. A list containing available data service queries displays.

h. Open CustomerProfile.ds (located in DataServices\CustomerManagement) and select the following methods:

- `getCustomerProfile()`
- `submitCustomerProfile()`

6. Click Add and then Finish.

Figure 17-5 Selecting Query Functions



7. After creating the ALDSP control, perform the following steps:

a. Open the `CustomerData.jcx` control in Source View.

b. Add a new function with the same signature as the `getCustomerProfile()` function and name it getCustomerProfileWithFilter.

c. Add the following parameter to the `getCustomerProfileWithFilter()` function:

```
FilterXQuery filter
```

d. After adding this parameter, the function signature will display as follows:

```
getCustomerProfileWithFilter (CustomerID String, filterXQuery filter)
```

17.2 Testing the Control and Retrieving Data

As with all data services, you should test functionality before you deploy the application.

Objectives

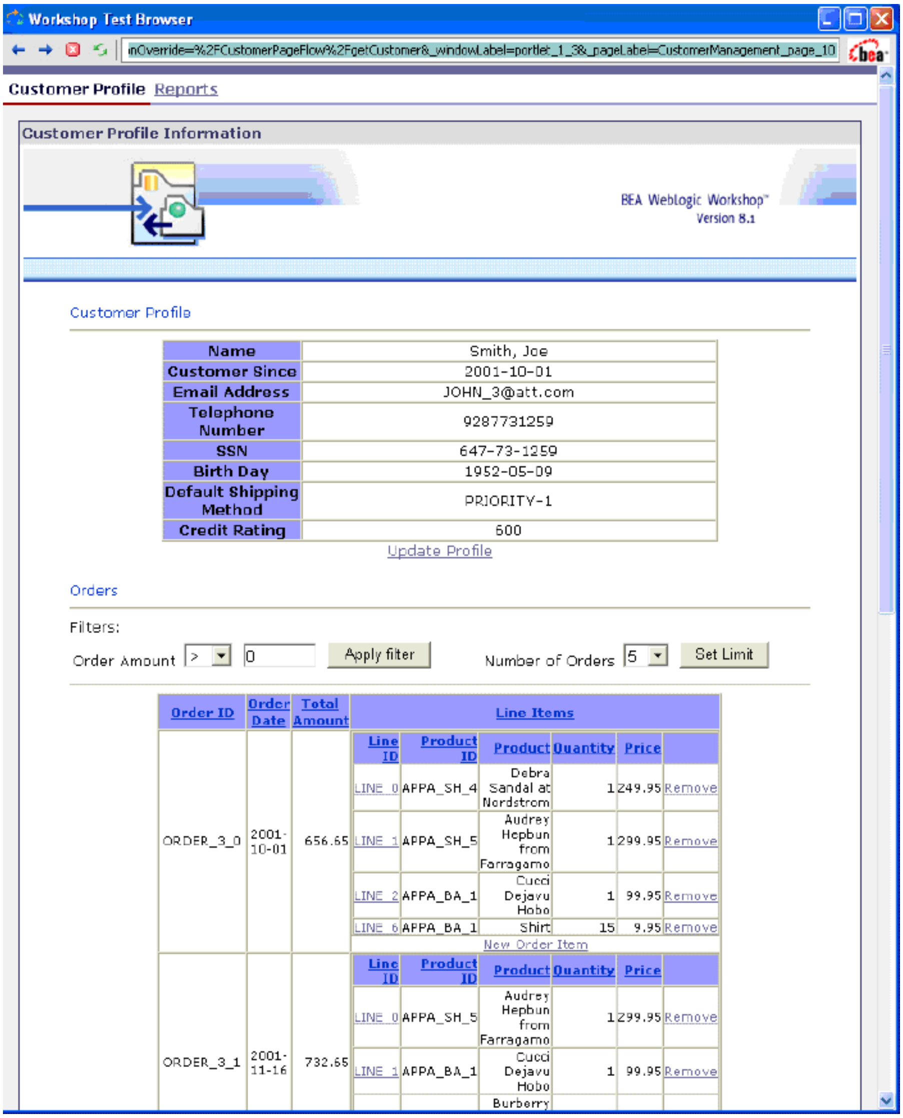
In this exercise, you will:

- Run the CustomerManagement.portal application.
- Retrieve data.
- Review the results.

Instructions

1. Open `CustomerManagement.portal`.
    a. Click the Start icon to open the Workshop Test Browser and run the portal application containing the CustomerManagementWebApp and the CustomerReport that were used in earlier lessons.
    b. Enter CUSTOMER3 in the Customer ID field and press Submit. The Customer Profile Information page opens.

Figure 17-6 Portal Access to Web Application Data



Workshop Test Browser
nOverride=%2FCustomerPageFlow%2FgetCustomer&_windowLabel=portlet_1_3&_pageLabel=CustomerManagement_page_10
Customer Profile
Reports
Customer Profile Information
BEA Weblogic Workshop™
Version 8.1

Customer Profile

Name	Smith, Joe
Customer Since	2001-10-01
Email Address	JOHN_3@att.com
Telephone Number	9287731259
SSN	647-73-1259
Birth Day	1952-05-09
Default Shipping Method	PRIORITY-1
Credit Rating	600

Update Profile

Orders

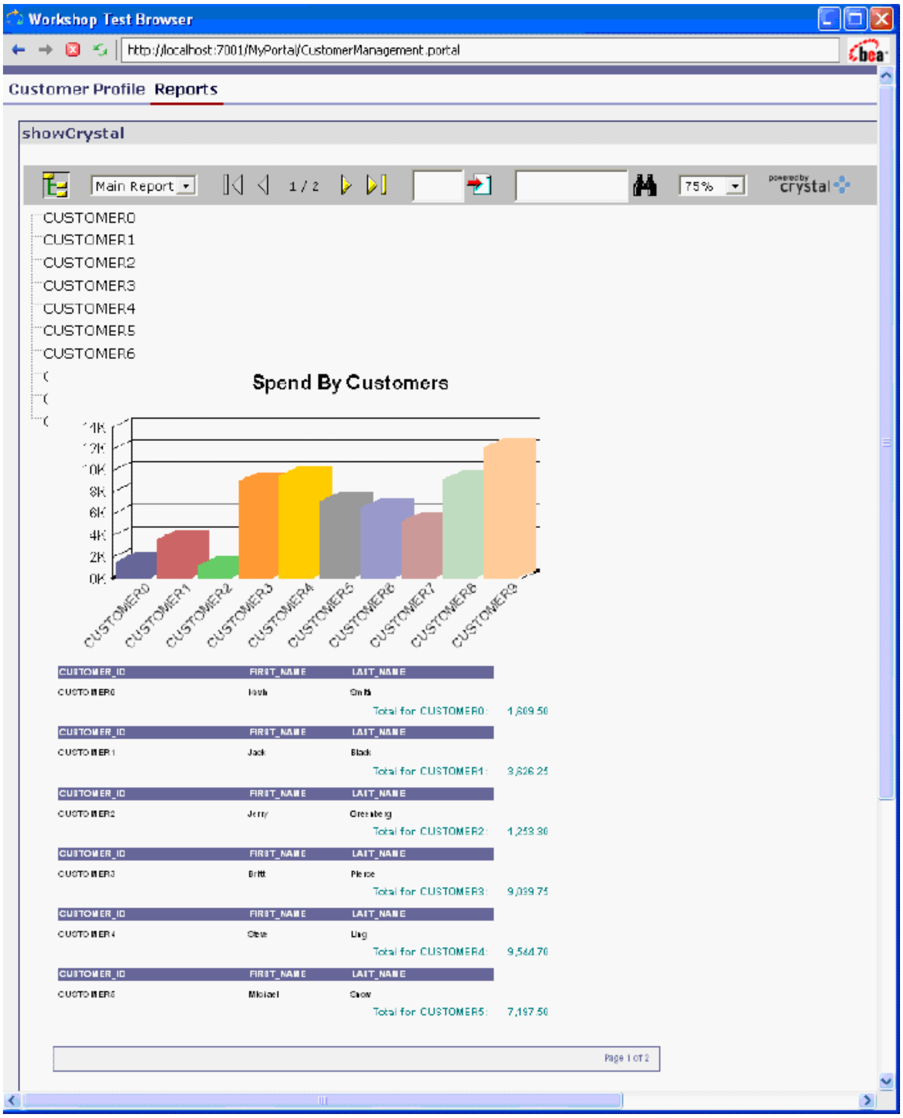
Filters:

Order Amount > 0 Apply filter Number of Orders 5 Set Limit

Order ID	Order Date	Total Amount	Line Items					
			Line ID	Product ID	Product	Quantity	Price	
ORDER_3_0	2001-10-01	656.65	LINE_0	APPA_SH_4	Debra Sandal at Nordstrom	1	249.95	Remove
			LINE_1	APPA_SH_5	Audrey Hepbun from Farragamo	1	299.95	Remove
			LINE_2	APPA_BA_1	Gucci Dejavu Hobo	1	99.95	Remove
			LINE_6	APPA_BA_1	Shirt	15	9.95	Remove
			New Order Item					
			Line ID	Product ID	Product	Quantity	Price	
ORDER_3_1	2001-11-16	732.65	LINE_0	APPA_SH_5	Audrey Hepbun from Farragamo	1	299.95	Remove
			LINE_1	APPA_BA_1	Gucci Dejavu Hobo	1	99.95	Remove
					Burberry			

c. Click the Reports link. For the Reports page, the first invocation may take a few moments before displaying.

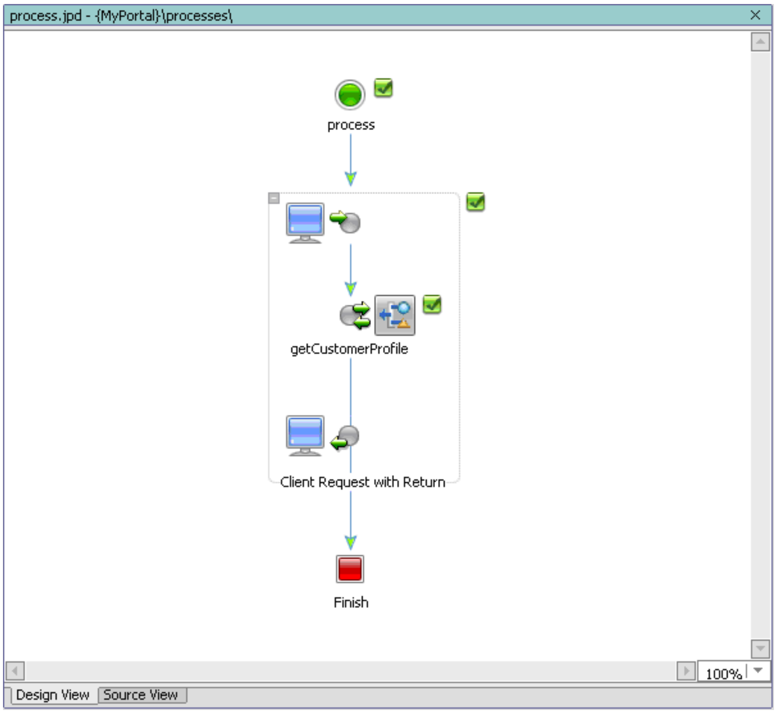
Figure 17-7 Portal Access to Crystal Reports Data



CUSTOMER_ID	FIRST_NAME	LAST_NAME		
CUSTOMER0	Kevin	Smith		
			Total for CUSTOMER0:	1,809.50
CUSTOMER1	Jack	Black		
			Total for CUSTOMER1:	3,826.25
CUSTOMER2	Jerry	Greenberg		
			Total for CUSTOMER2:	1,253.30
CUSTOMER3	Britt	Pierce		
			Total for CUSTOMER3:	9,039.75
CUSTOMER4	Steve	Ling		
			Total for CUSTOMER4:	9,544.70
CUSTOMER5	Michael	Snow		
			Total for CUSTOMER5:	7,197.50

d. Open the `process.jpd` file, located in the MyPortal\processes folder. You will see the Design View of the process definition that accepts a CUSTOMER_ID String, invokes the Data Service Control, and returns the customer information in an XML document.

Figure 17-8 Design View of process.jpd File

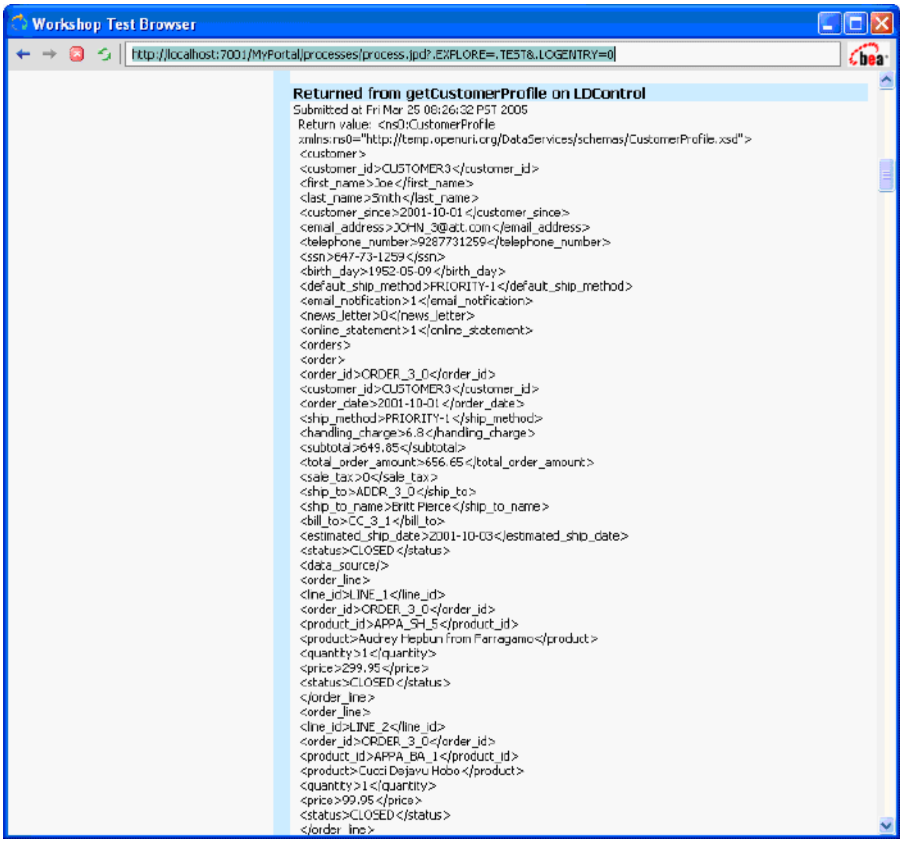

process.jpd - {MyPortal}\processes\
process
getCustomerProfile
Client Request with Return
Finish
100%
Design View
Source View

e. Click the Start icon to test the process definition.

f. Enter CUSTOMER3 in the Customer ID field and then click clientRequestwithReturn.

g. Scroll through the page to view customer information included in the "Returned from getCustomerProfile on LDControl" section.

Figure 17-9 Business Process View of Customer Data


Workshop Test Browser
http://localhost:7001/MyPortal/processes/process.jpd?.EXPLORE=.TEST&.LOGENTRY=0

Returned from getCustomerProfile on LDControl

```
Submitted at Fri Mar 25 08:26:32 PST 2005
 Return value:  <ns0:CustomerProfile
 xmlns:ns0="http://temp.openuri.org/DataServices/schemas/CustomerProfile.xsd">
 <customer>
 <customer_id>CUSTOMER3</customer_id>
 <first_name>Joe</first_name>
 <last_name>Smith</last_name>
 <customer_since>2001-10-01</customer_since>
 <email_address>JOHN_3@att.com</email_address>
 <telephone_number>9287731259</telephone_number>
 <ssn>647-73-1259</ssn>
 <birth_day>1952-05-09</birth_day>
 <default_ship_method>PRIORITY-1</default_ship_method>
 <email_notification>1</email_notification>
 <news_letter>0</news_letter>
 <online_statement>1</online_statement>
 <orders>
 <order>
 <order_id>ORDER_3_0</order_id>
 <customer_id>CUSTOMER3</customer_id>
 <order_date>2001-10-01</order_date>
 <ship_method>PRIORITY-1</ship_method>
 <handling_charge>6.8</handling_charge>
 <subtotal>649.85</subtotal>
 <total_order_amount>656.65</total_order_amount>
 <sale_tax>0</sale_tax>
 <ship_to>ADDR_3_0</ship_to>
 <ship_to_name>Britt Pierce</ship_to_name>
 <bill_to>CC_3_1</bill_to>
 <estimated_ship_date>2001-10-03</estimated_ship_date>
 <status>CLOSED</status>
 <data_source/>
 <order_line>
 <line_id>LINE_1</line_id>
 <order_id>ORDER_3_0</order_id>
 <product_id>APPA_SH_5</product_id>
 <product>Audrey Hepbun from Farragamo</product>
 <quantity>1</quantity>
 <price>299.95</price>
 <status>CLOSED</status>
 </order_line>
 <order_line>
 <line_id>LINE_2</line_id>
 <order_id>ORDER_3_0</order_id>
 <product_id>APPA_BA_1</product_id>
 <product>Cucci Dejavu Hobo</product>
 <quantity>1</quantity>
 <price>99.95</price>
 <status>CLOSED</status>
 </order_line>
```

Lesson Summary

In this lesson you learned how to:

- Import a WebLogic Portal project that contains portals and business processes.
- Install the Data Service Control in the project, thereby making data services available from the portal and business processes.
- Recognize how a Data Service Control is used from a portal and business process.

Building XQueries in XQuery Editor View

In concrete terms, a data service is simply a file that contains XML Query (XQuery) instructions for retrieving, aggregating, and transforming data. Essentially you create a query function by:

- Integrating physical and logical data sources into the query.
- Mapping data sources to the data service's Return type.
- Creating XQuery statements that include conditions, parameters, functions, and expressions.

You can also modify the Return type, either within XQuery Editor View or using an external tool.

In this lesson, you will use XQuery Editor View to develop a variety of XQuery instructions.

Objectives

After completing this lesson, you will be able to:

- Use the graphical XQuery Editor View to create parameterized, string, and date functions; outer joins, aggregate, and order by and constant expressions.
- Use the XQuery Function Palette to add built-in XQuery functions to a query.

Overview

XQuery Editor View provides a graphical, drag-and-drop approach to constructing queries. Using XQuery Editor View, you can:

- View and modify the data service's Return type, whose shape is defined by the data service's XML Type.
- View, add, modify, and delete the function calls from other physical and logical data services that define which data source(s) will be queried.
- View, add, and delete the source-to-target mappings that define which data will be made available to consuming applications.
- View, add, modify, and delete the parameters, expressions, and conditions that define how the data will be processed.

Changes that you make in XQuery Editor View are immediately reflected in Source View. Similarly, changes you make in Source View will be immediately effective in XQuery Editor View.

18.1 Importing Schemas for Query Development

To simplify development time in this lesson you will use ready-made schemas that define a data service's Return type.

Objectives

In this exercise, you will:

- Create a folder to organize all the queries that you will create in this lesson and the next.
- Import the schemas that you will use in those queries.

Instructions

1. Create a new folder in the `DataServices` project folder, and name it `MyQueries`.
   a. Right-click the `MyQueries` folder and choose Import.
   b. Navigate to `<beahome>\weblogic81\samples\LiquidData\EvalGuide\MyQueries`, select the `schemas` folder, and click Import. This will automatically create a folder named `schemas`, and appropriate `.xsd` files, within the `MyQueries` directory. These `.xsd` files will be used to determine the Return type for all queries developed in this lesson.

18.2 Creating Source-to-Target Mappings

Every function within a logical data service includes source-to-target mappings that define what results will be returned by the function. As described in Part I, there are several types of mappings:

- A simple mapping means that you are mapping simple source node elements to simple elements in the Return type one at a time. You can create a simple mapping by dragging and dropping any element from the source node to its corresponding target element in the Return type. Optional Return type elements do not need to be mapped; otherwise elements in the Return type need to be mapped in order for your query to run.
- An induced mapping means that a complex element is mapped to a complex element in the Return type. In this gesture the top level complex element in the Return type is ignored (source node name need not match). The editor automatically then maps any child elements (complex or simple) that are an exact match for source node elements.
- An overwrite mapping replaces a Result type element and all its children (if any) with the source node elements. As an example of the general steps needed to create an overwrite mapping, you would press <Ctrl>, then drag and drop the source node's complex element onto the corresponding element in the Result type. The entire source node's complex element is brought to the Result type, where it completely replaces the target element with the source element.
- An append mapping adds a simple or complex element (and any children or attributes) as a child of the specified element in the Return type. To create an append mapping, select the source element, then press <Ctrl>+<Shift> while dragging and dropping the source node's element onto the element in the Return type that you want to be the parent of the new element(s).

Alternatively, if you simply want to add a child element to a Return type, you can drag a source element to a complex element in your Return type. The element will be added as a child of the complex element and mapped accordingly.

Objectives

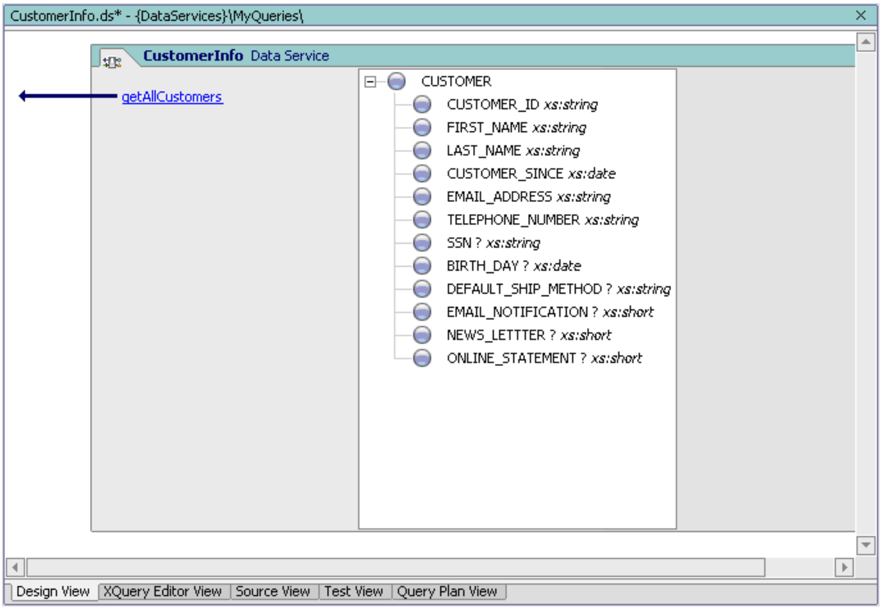
In this exercise, you will:

- Create four types of mappings.
- Review the results.

Instructions

1. Right-click the `MyQueries` folder, choose New → Data Service, and use `CustomerInfo.ds` in the Name field.
2. In Design View, associate the CustomerInfo data service with the `CUSTOMER.xsd` schema. The schema is located in `MyQueries\schemas`.
3. Add a new function to the CustomerInfo data service and name it getAllCustomers.

Figure 18-1 Design View of CustomerInfo Data Service

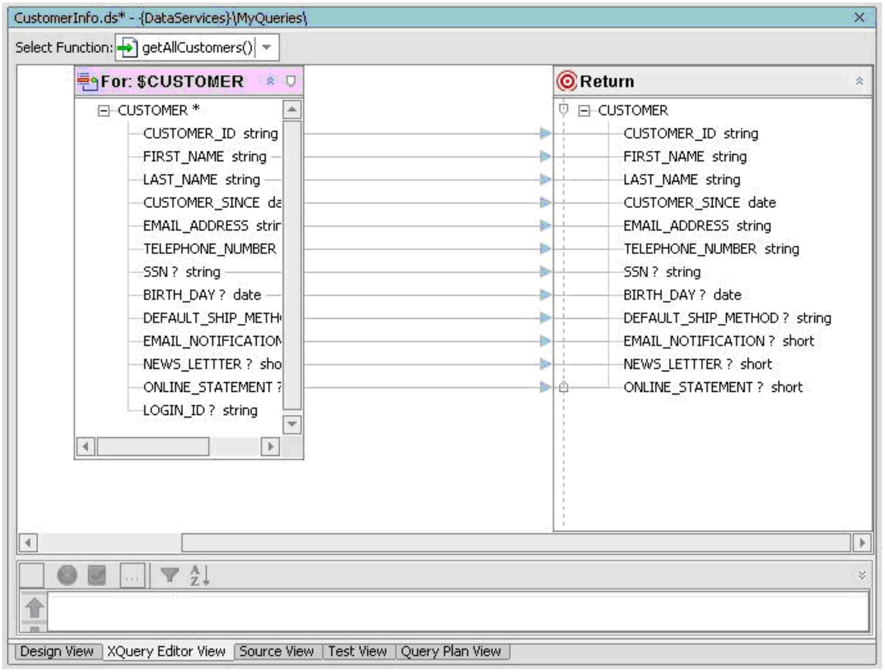


4. Click the `getAllCustomers()` function to open XQuery Editor View.
5. Add a for node to the work area by completing the following steps:
   a. In the Data Services Palette, open the `CUSTOMER.ds` folder, located in `DataServices\CustomerDB`.
   b. Drag and drop `CUSTOMER()` into XQuery Editor View. This creates a `For:$CUSTOMER source` node.

6. Create a simple mapping. Drag and drop each element in the CUSTOMER source node onto the corresponding element in the Return type.

   Note: You do not need to map the LOGIN_ID element.

Figure 18-2 Simple Mapping



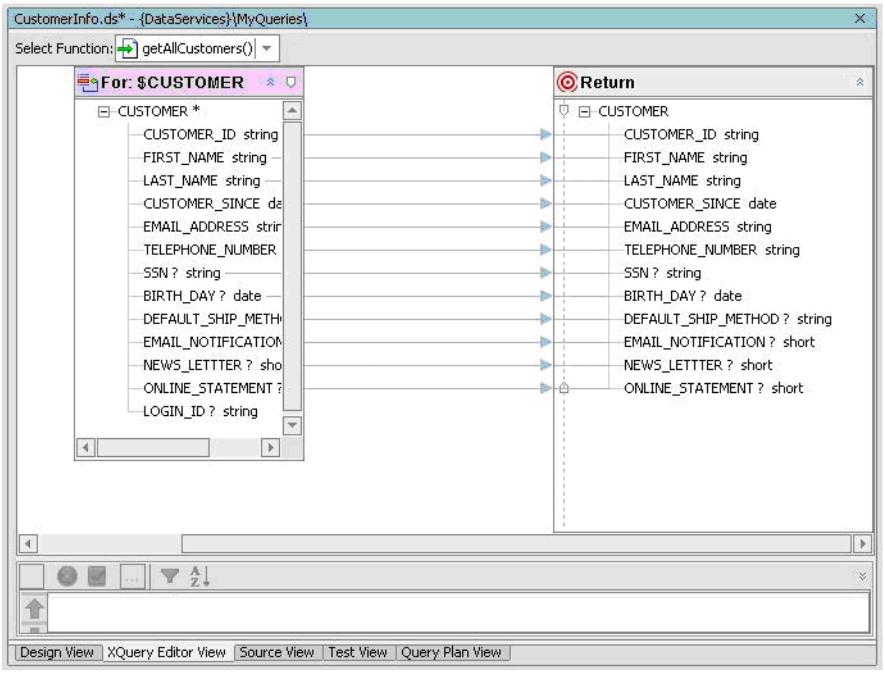
7. Create an induced mapping, by completing the following steps:

   a. Delete all the simple mappings. (Right-click a map line and select Delete from the pop-up menu.)

   b. Drag and drop the CUSTOMER* element (source node) onto the CUSTOMER element in the Return type.

   Notice that the mappings are automatically generated for each element, because the source and target element names are the same.

Figure 18-3 Induced Mapping



8. Create an overwrite mapping, by completing the following steps:

   a. In the Return type right-click the CUSTOMER element and choose Add Child Element.

   b. Double-click the NewChildElement, enter Addresses, and press Enter.

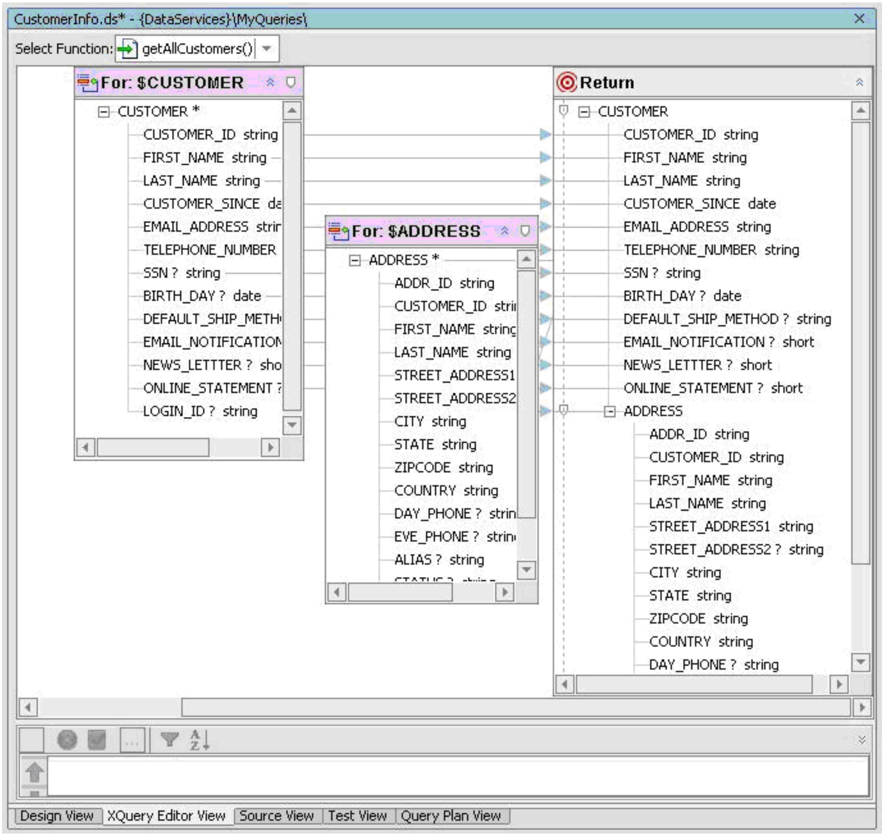
   c. In the Data Services Palette, open the `ADDRESS.ds` icon, which is located in the `DataServices\CustomerDB` folder.

   d. Drag and drop `ADDRESS()` into XQuery Editor View.

   e. Press Ctrl, and then drag and drop ADDRESS* element (source node) onto the Addresses element in the Return type.

   Notice that the entire complex ADDRESS* element is brought to the target, where it overwrites the element, instead of adding it as a child.

Figure 18-4 Overwrite Mapping



18.3 Creating a Basic Parameterized Function

A parameterized query lets you filter returned data based on specific criteria, such as a particular order number, customer name, or customer number.

Objectives

In this exercise, you will:

- Create a parameterized function that returns all orders for a particular customer.
- Test the function.

- Review the XQuery source code.

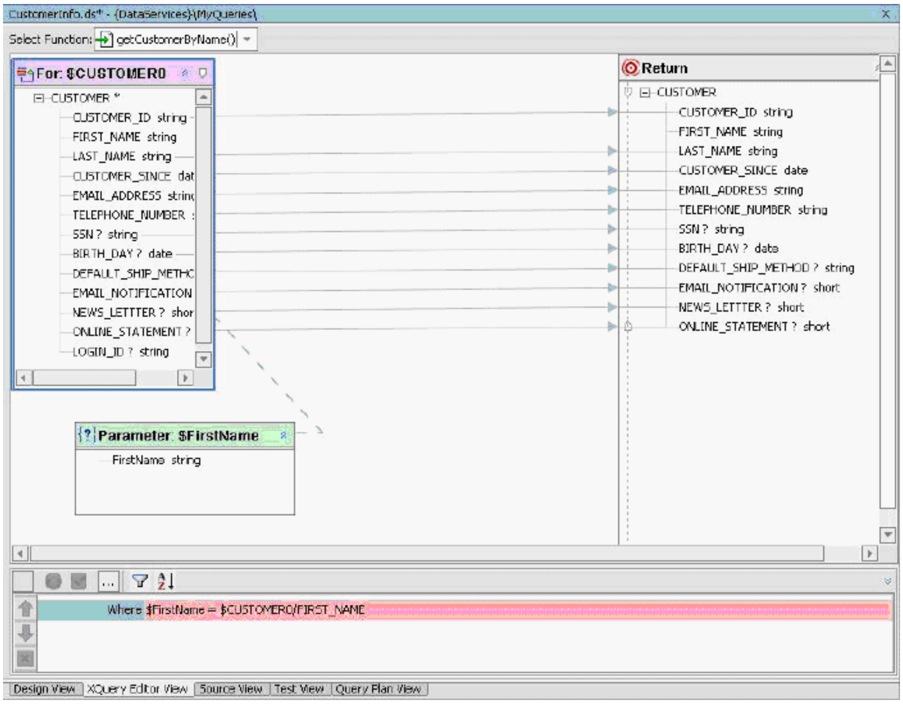
Instructions

In Design View: Add a new function to the CustomerInfo data service and name it getCustomerByName.

1. Click `getCustomerByName()` to open XQuery Editor View for that function.
2. Add a for node, by completing the following steps:
   a. In the Data Services Palette, open the `CUSTOMER.ds` folder, which is located in the `DataServices\CustomerDB` folder.
   b. Drag and drop `CUSTOMER()` into XQuery Editor View. This creates a `For:$CUSTOMER` source node.
3. Create an induced mapping. Drag and drop the CUSTOMER* element (source node) onto the CUSTOMER element in the Return type.
4. Add a parameter, by completing the following steps:
   a. Right-click an empty spot in XQuery Editor View.
   b. Choose Add Parameter.
   c. Enter FirstName in the Parameter Name field.
   d. Select xs:string as the Primitive Type.
   e. Click OK. (You will need to move the nodes until all are visible because the new parameter node may be placed behind the CUSTOMER node.)
5. Add a where clause, by completing the following steps:
   a. Drag and drop the parameter's string element onto FIRST_NAME element (source node). Make sure that you release the mouse button when the FIRST_NAME element is highlighted. This action creates a filter for the FIRST_NAME element based on the parameter that is passed to the function.
   b. Confirm that the where clause is correctly set by clicking the $CUSTOMER source node's header. The Expression Editor will open and you should see the following where clause:

```
$FirstName = $CUSTOMER0/FIRST_NAME
```

Figure 18-5 First Name Parameter and WHERE Clause



Where $FirstName = $CUSTOMER0/FIRST_NAME

6. Add a second where clause, by completing the following steps:
   a. Add a new parameter, entering LastName, and selecting xs:string as the Primitive Type.
   b. Click the $CUSTOMER node's header. The Expression Editor opens.
   c. Triple-click inside the where field and place your cursor at the very end, after FIRST_NAME.
   d. Select the “and” logical conjunction from the pop-up operator list (the “...” icon). You can now define the where clause to filter data by last name.

      Note: An alternative method is to simply enter “and” in the field.
   e. Click the string element in the second parameter. The variable name $LastName appears at the end of the where clause.
   f. Choose eq: Compare Single Values from the popup operator list.

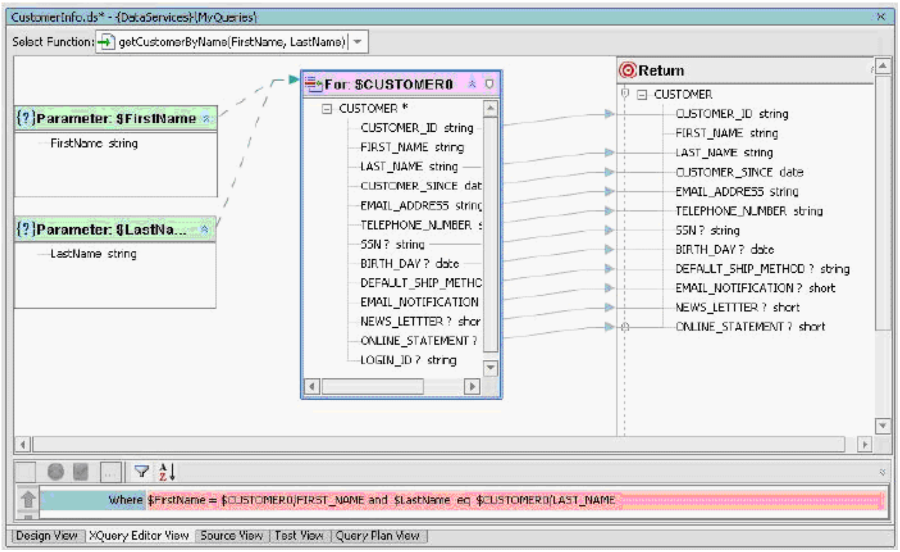
      Note: An alternative method is to simply enter eq in the field.

g. Click the LAST_NAME element in the For:$CUSTOMER node. You should see the following in the where clause field:

```
$FirstName = $CUSTOMER/FIRST_NAME and $LastName = $CUSTOMER/LAST_NAME
```

h. Click the green check button to accept the changes.

Figure 18-6 Query Editor View of Parameterized Query



Where $FirstName = $CUSTOMER0/FIRST_NAME and $LastName eq $CUSTOMER0/LAST_NAME

7. Test the function, by completing the following steps:

a. Open `CustomerInfo.ds` in Test View.

b. Select `getCustomerByName(FirstName, LastName)` from the drop-down list.

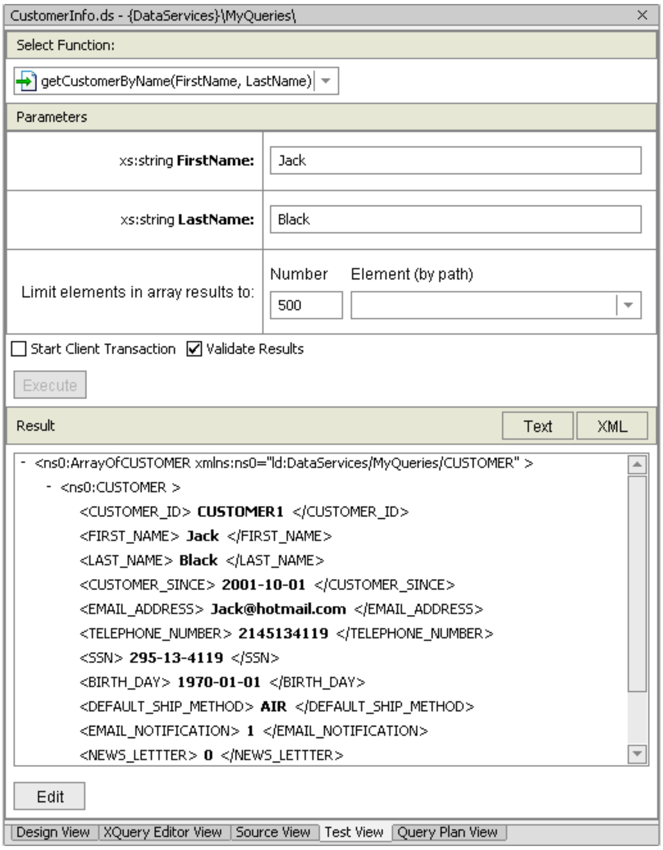
c. Enter Jack in $FirstName field.

d. Enter Black in the $LastName field.

e. Click Execute.

Confirm the results, which should be as displayed in Figure 18-7.

Figure 18-7 Parameterized Query Results



getCustomerByName(FirstName, LastName)

xs:string **FirstName:** Jack

xs:string **LastName:** Black

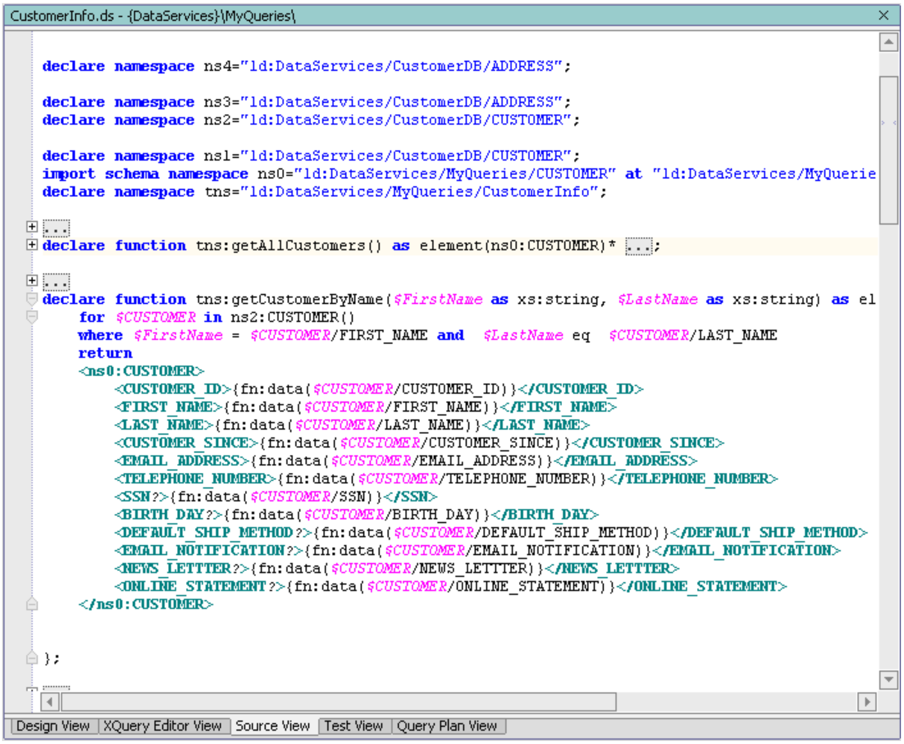
Limit elements in array results to: Number 500 Element (by path)

```
- <ns0:ArrayOfCUSTOMER xmlns:ns0="ld:DataServices/MyQueries/CUSTOMER" >
  - <ns0:CUSTOMER >
      <CUSTOMER_ID> CUSTOMER1 </CUSTOMER_ID>
      <FIRST_NAME> Jack </FIRST_NAME>
      <LAST_NAME> Black </LAST_NAME>
      <CUSTOMER_SINCE> 2001-10-01 </CUSTOMER_SINCE>
      <EMAIL_ADDRESS> Jack@hotmail.com </EMAIL_ADDRESS>
      <TELEPHONE_NUMBER> 2145134119 </TELEPHONE_NUMBER>
      <SSN> 295-13-4119 </SSN>
      <BIRTH_DAY> 1970-01-01 </BIRTH_DAY>
      <DEFAULT_SHIP_METHOD> AIR </DEFAULT_SHIP_METHOD>
      <EMAIL_NOTIFICATION> 1 </EMAIL_NOTIFICATION>
      <NEWS_LETTTER> 0 </NEWS_LETTTER>
```

8. Open `CustomerInfo.ds` in Source View to view the generated XQuery. The query should be similar to that displayed in Figure 18-8.

   Note: The automatic namespace assignments may not match.

Figure 18-8 Parameterized Function Source Code



```
declare namespace ns4="ld:DataServices/CustomerDB/ADDRESS";

declare namespace ns3="ld:DataServices/CustomerDB/ADDRESS";
declare namespace ns2="ld:DataServices/CustomerDB/CUSTOMER";

declare namespace ns1="ld:DataServices/CustomerDB/CUSTOMER";
import schema namespace ns0="ld:DataServices/MyQueries/CUSTOMER" at "ld:DataServices/MyQuerie
declare namespace tns="ld:DataServices/MyQueries/CustomerInfo";

...
declare function tns:getAllCustomers() as element(ns0:CUSTOMER)* ...;

...
declare function tns:getCustomerByName($FirstName as xs:string, $LastName as xs:string) as el
    for $CUSTOMER in ns2:CUSTOMER()
    where $FirstName = $CUSTOMER/FIRST_NAME and  $LastName eq  $CUSTOMER/LAST_NAME
    return
    <ns0:CUSTOMER>
        <CUSTOMER_ID>{fn:data($CUSTOMER/CUSTOMER_ID)}</CUSTOMER_ID>
        <FIRST_NAME>{fn:data($CUSTOMER/FIRST_NAME)}</FIRST_NAME>
        <LAST_NAME>{fn:data($CUSTOMER/LAST_NAME)}</LAST_NAME>
        <CUSTOMER_SINCE>{fn:data($CUSTOMER/CUSTOMER_SINCE)}</CUSTOMER_SINCE>
        <EMAIL_ADDRESS>{fn:data($CUSTOMER/EMAIL_ADDRESS)}</EMAIL_ADDRESS>
        <TELEPHONE_NUMBER>{fn:data($CUSTOMER/TELEPHONE_NUMBER)}</TELEPHONE_NUMBER>
        <SSN?>{fn:data($CUSTOMER/SSN)}</SSN>
        <BIRTH_DAY?>{fn:data($CUSTOMER/BIRTH_DAY)}</BIRTH_DAY>
        <DEFAULT_SHIP_METHOD?>{fn:data($CUSTOMER/DEFAULT_SHIP_METHOD)}</DEFAULT_SHIP_METHOD>
        <EMAIL_NOTIFICATION?>{fn:data($CUSTOMER/EMAIL_NOTIFICATION)}</EMAIL_NOTIFICATION>
        <NEWS_LETTTER?>{fn:data($CUSTOMER/NEWS_LETTTER)}</NEWS_LETTTER>
        <ONLINE_STATEMENT?>{fn:data($CUSTOMER/ONLINE_STATEMENT)}</ONLINE_STATEMENT>
    </ns0:CUSTOMER>


};
```

18.4: Creating a String Function with a Built-In XQuery Function

The XQuery language provides more than 100 functions. BEA provides some additional, special purpose functions. In this exercise, you will build a query that uses the built-in `XQuery startWith()` function to create business logic sufficient to retrieve records based on an OR condition.

Objectives

In this exercise, you will:

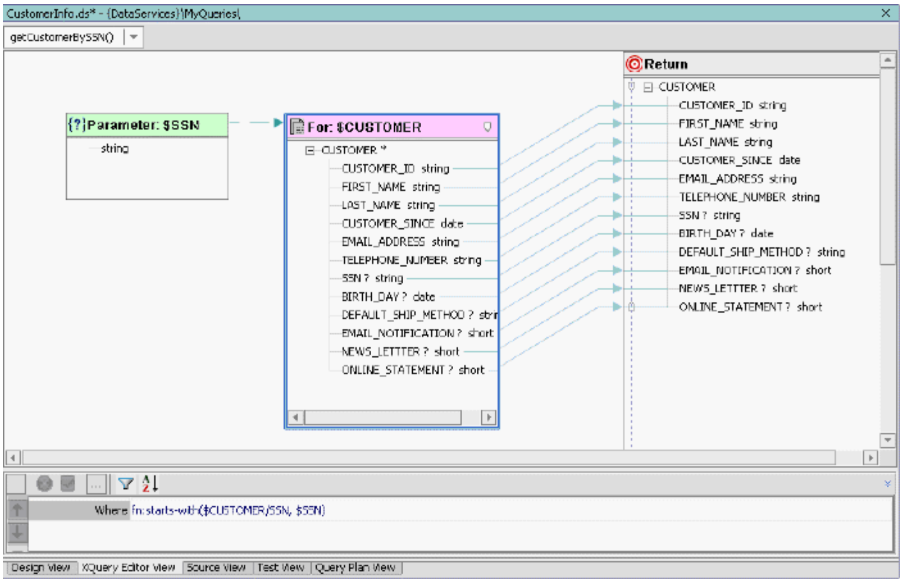
- Create a string function that will find customers by their social security number.

- Test the function.
- Review the XQuery source code.

Instructions

1. Add a new function to the CustomerInfo data service and name it getCustomerBySSN.
2. Click `getCustomerBySSN()` to open XQuery Editor View to that function.
3. Add a for clause, by completing the following steps:
   a. In the Data Services Palette, open the `CUSTOMER.ds` folder, which is located in `DataServices\CustomerDB`.
   b. Drag and drop `CUSTOMER()` into XQuery Editor View. This creates a For:$CUSTOMER node.
4. 4.Create an induced map. Drag and drop the CUSTOMER* element (source) onto the CUSTOMER element in the Return type.
5. Add a new parameter, entering SSN as the Parameter Name, and selecting xs:string as the Primitive Type.
6. Add a where clause that uses a built-in XQuery function, by completing the following steps:
   a. Click the $CUSTOMER node's header. The Expression Editor opens.
   b. Click the Add Where Clause icon.
   c. In XQuery Function Palette, expand the String Functions folder.
   d. Drag and drop the following function into the where clause field.
   ```
   fn:starts-with($arg1 as xs:string?, $arg2 as xs:string?) as xs:boolean
   ```
   e. Confirm that the where clause now includes the following built-in function:
   ```
    fn:starts-with($arg1, $arg2)
   ```
   f. Edit the where clause, so that it reads as follows:
   ```
   fn:starts-with($CUSTOMER/SSN, $SSN)
   ```
   g. Click the green check button to accept the changes.

Figure 18-9 Built-In Function Where Clause



Where fn:starts-with($CUSTOMER/SSN, $SSN)

7. Test the function, by completing the following steps:

   a. Open `CustomerInfo.ds` in Test View.

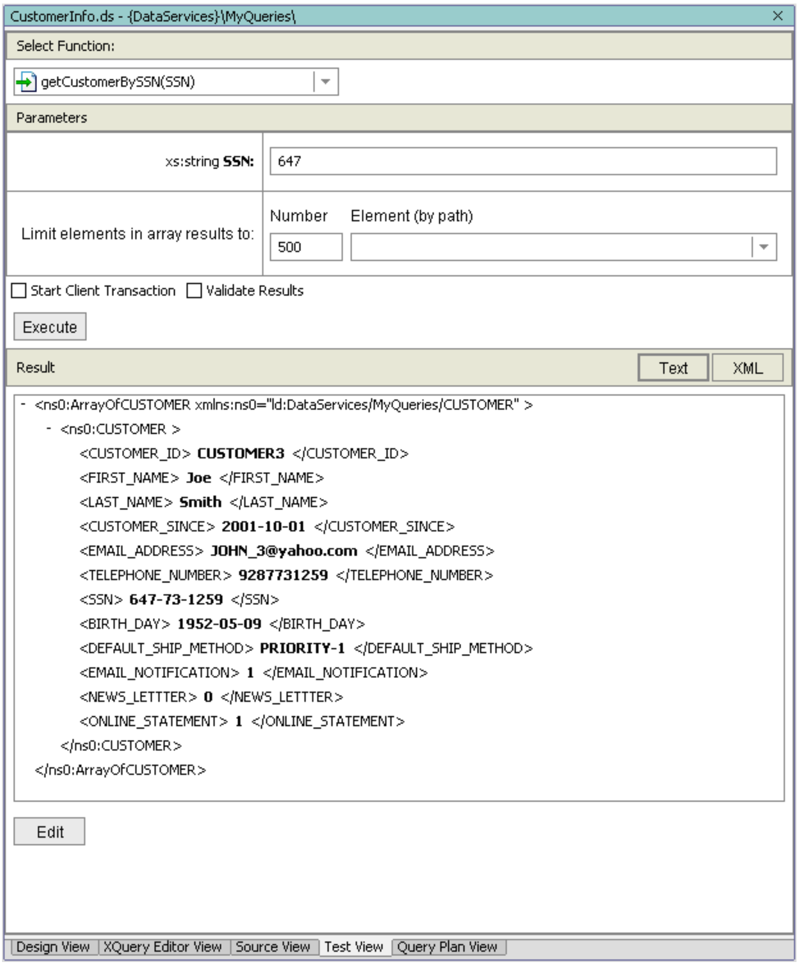
   b. Select `getCustomerBySSN()` from the Function drop-down list.

   c. Enter 647 in the xs:string SSN field.

   d. Click Execute.

   e. Confirm the results, which should be as displayed in Figure 18-10.

Figure 18-10 Built-In Function Test Results



xs:string **SSN:** 647

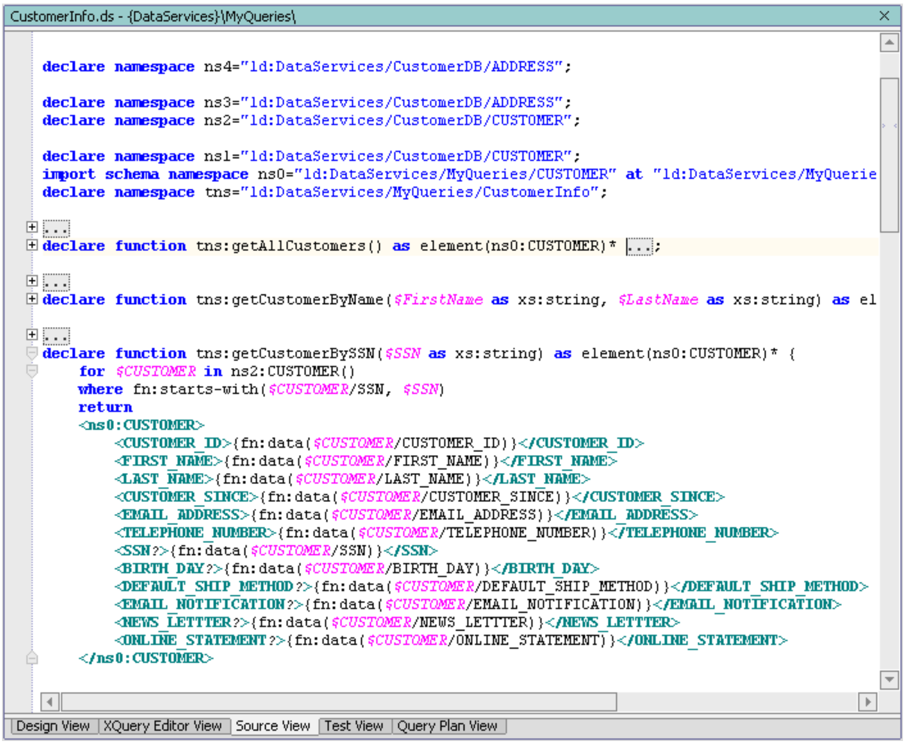
Limit elements in array results to: Number 500 Element (by path)

```
- <ns0:ArrayOfCUSTOMER xmlns:ns0="ld:DataServices/MyQueries/CUSTOMER" >
  - <ns0:CUSTOMER >
      <CUSTOMER_ID> CUSTOMER3 </CUSTOMER_ID>
      <FIRST_NAME> Joe </FIRST_NAME>
      <LAST_NAME> Smith </LAST_NAME>
      <CUSTOMER_SINCE> 2001-10-01 </CUSTOMER_SINCE>
      <EMAIL_ADDRESS> JOHN_3@yahoo.com </EMAIL_ADDRESS>
      <TELEPHONE_NUMBER> 9287731259 </TELEPHONE_NUMBER>
      <SSN> 647-73-1259 </SSN>
      <BIRTH_DAY> 1952-05-09 </BIRTH_DAY>
      <DEFAULT_SHIP_METHOD> PRIORITY-1 </DEFAULT_SHIP_METHOD>
      <EMAIL_NOTIFICATION> 1 </EMAIL_NOTIFICATION>
      <NEWS_LETTTER> 0 </NEWS_LETTTER>
      <ONLINE_STATEMENT> 1 </ONLINE_STATEMENT>
    </ns0:CUSTOMER>
  </ns0:ArrayOfCUSTOMER>
```

8. Open CustomerInfo.ds in Source View to view the generated XQuery. The query should be similar to that displayed in Figure 18-11.

   Note: The automatic namespace assignments may not match.

Figure 18-11 Source View of Built-In String Function



CustomerInfo.ds - {DataServices}\MyQueries\
Design View | XQuery Editor View | Source View | Test View | Query Plan View

```
declare namespace ns4="ld:DataServices/CustomerDB/ADDRESS";

declare namespace ns3="ld:DataServices/CustomerDB/ADDRESS";
declare namespace ns2="ld:DataServices/CustomerDB/CUSTOMER";

declare namespace ns1="ld:DataServices/CustomerDB/CUSTOMER";
import schema namespace ns0="ld:DataServices/MyQueries/CUSTOMER" at "ld:DataServices/MyQuerie
declare namespace tns="ld:DataServices/MyQueries/CustomerInfo";

...
declare function tns:getAllCustomers() as element(ns0:CUSTOMER)* ...;

...
declare function tns:getCustomerByName($FirstName as xs:string, $LastName as xs:string) as el

...
declare function tns:getCustomerBySSN($SSN as xs:string) as element(ns0:CUSTOMER)* {
    for $CUSTOMER in ns2:CUSTOMER()
    where fn:starts-with($CUSTOMER/SSN, $SSN)
    return
    <ns0:CUSTOMER>
        <CUSTOMER_ID>{fn:data($CUSTOMER/CUSTOMER_ID)}</CUSTOMER_ID>
        <FIRST_NAME>{fn:data($CUSTOMER/FIRST_NAME)}</FIRST_NAME>
        <LAST_NAME>{fn:data($CUSTOMER/LAST_NAME)}</LAST_NAME>
        <CUSTOMER_SINCE>{fn:data($CUSTOMER/CUSTOMER_SINCE)}</CUSTOMER_SINCE>
        <EMAIL_ADDRESS>{fn:data($CUSTOMER/EMAIL_ADDRESS)}</EMAIL_ADDRESS>
        <TELEPHONE_NUMBER>{fn:data($CUSTOMER/TELEPHONE_NUMBER)}</TELEPHONE_NUMBER>
        <SSN?>{fn:data($CUSTOMER/SSN)}</SSN>
        <BIRTH_DAY?>{fn:data($CUSTOMER/BIRTH_DAY)}</BIRTH_DAY>
        <DEFAULT_SHIP_METHOD?>{fn:data($CUSTOMER/DEFAULT_SHIP_METHOD)}</DEFAULT_SHIP_METHOD>
        <EMAIL_NOTIFICATION?>{fn:data($CUSTOMER/EMAIL_NOTIFICATION)}</EMAIL_NOTIFICATION>
        <NEWS_LETTTER?>{fn:data($CUSTOMER/NEWS_LETTTER)}</NEWS_LETTTER>
        <ONLINE_STATEMENT?>{fn:data($CUSTOMER/ONLINE_STATEMENT)}</ONLINE_STATEMENT>
    </ns0:CUSTOMER>
```

18.5: Creating a Date Function

A date function lets you retrieve data based on date parameters.

Objectives

In this exercise, you will:

- Create a date function that will find customers by the year that they were born.
- Test the function.
- Review the XQuery source code.

Instructions

1. Add a new function to the CustomerInfo data service and name it getCustomerByBirthYear.
2. Click `getCustomerByBirthYear()` to open XQuery Editor View to that function.
3. Add a for clause, by completing the following steps:
   a. In the Data Services Palette, open the `CUSTOMER.ds` folder, which is located in `DataServices\CustomerDB`.
   b. Drag and drop `CUSTOMER()` into XQuery Editor View. This creates a for node for the `CUSTOMER()` function.
4. Create an induced mapping. Drag and drop the CUSTOMER* element (source) onto the CUSTOMER element (Return).
5. Create a new parameter, enter BirthYear as the Parameter Name, and select xs:integer as the Primitive Type.
6. Add a where clause, by completing the following steps:
   a. Click the $CUSTOMER node's header. The Expression Editor opens.
   b. Click the Add Where Clause icon.
   c. In XQuery Function Palette, expand the Duration, Date, and Time Functions folder.
   d. Drag and drop the built-in following function into the where clause field.

   ```
   fn:year-from-date($arg as xs:date?) as xs:integer?
   ```

   e. Confirm that the where clause is as follows:

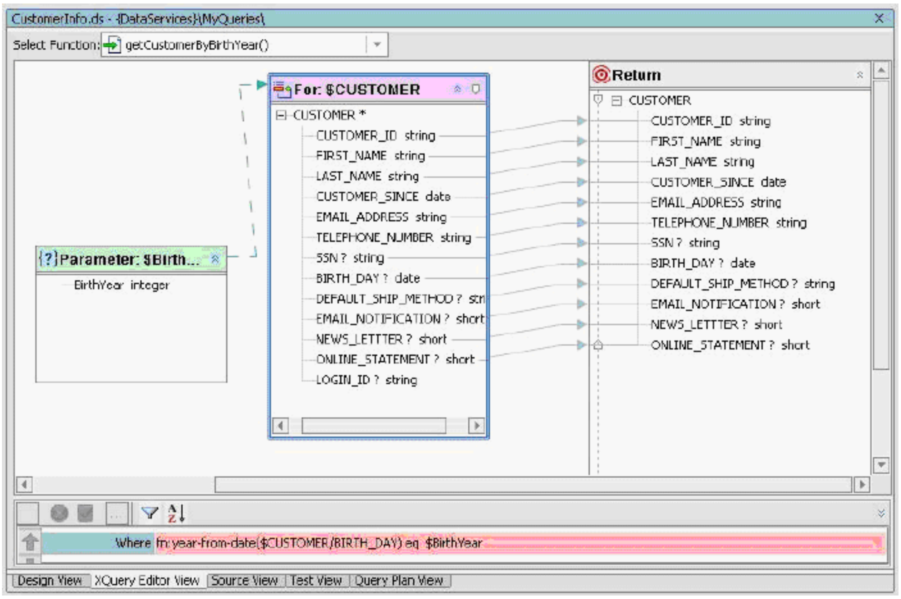
   ```
   fn:year-from-date($arg)
   ```

   f. Edit the built-in function, so that it reads as:

   ```
   fn:year-from-date($CUSTOMER/BIRTH_DAY) eq $BirthYear
   ```

   g. Click the green check button to accept the changes.

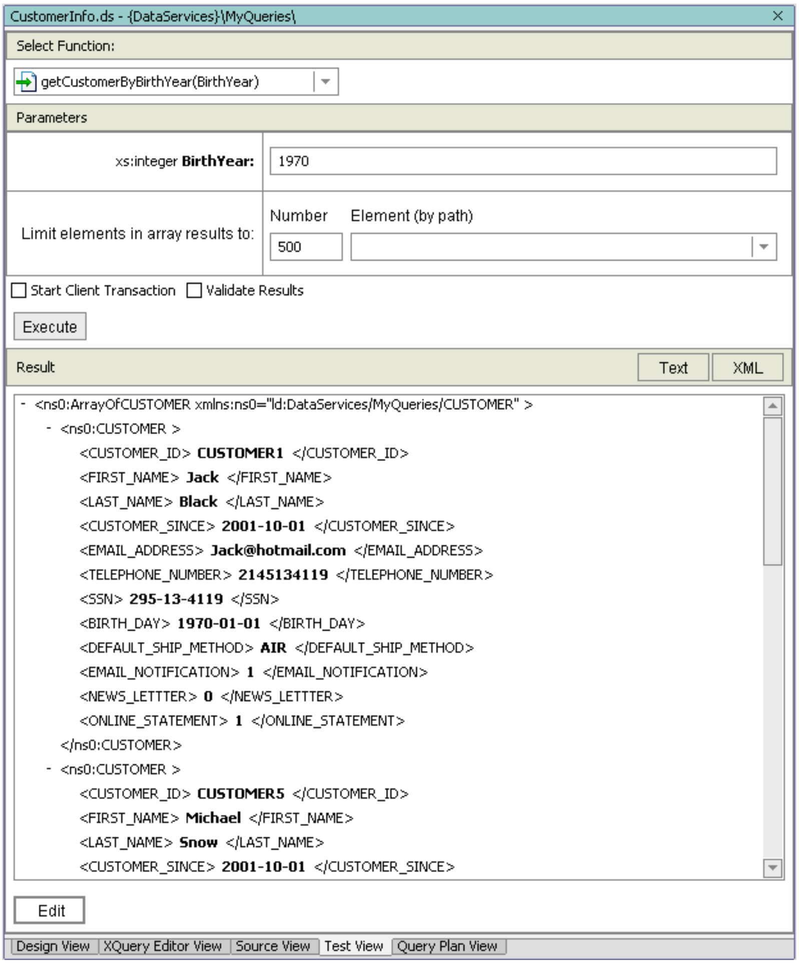
Figure 18-12 Where Clause Using a Built-In Date Function



Where fn:year-from-date($CUSTOMER/BIRTH_DAY) eq $BirthYear

7. Test the function, by completing the following steps:
   a. Open `CustomerInfo.ds` in Test View.
   b. Select `getCustomerByBirthYear()` from the function drop-down list.
   c. Enter 1970 in the $arg0 field.
   d. Click Execute.
   e. Confirm the results, which should be as displayed in Figure 18-13. There should be five customer profiles returned.

Figure 18-13 Date Function Test Results



xs:integer **BirthYear:** 1970

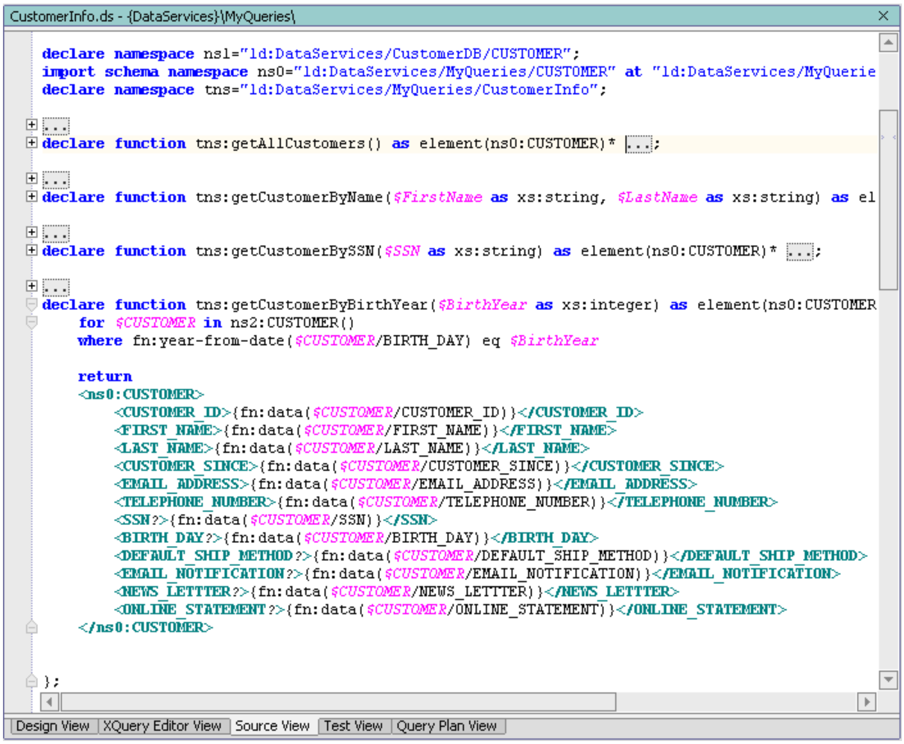
Limit elements in array results to: Number 500 Element (by path)

```
- <ns0:ArrayOfCUSTOMER xmlns:ns0="ld:DataServices/MyQueries/CUSTOMER" >
  - <ns0:CUSTOMER >
      <CUSTOMER_ID> CUSTOMER1 </CUSTOMER_ID>
      <FIRST_NAME> Jack </FIRST_NAME>
      <LAST_NAME> Black </LAST_NAME>
      <CUSTOMER_SINCE> 2001-10-01 </CUSTOMER_SINCE>
      <EMAIL_ADDRESS> Jack@hotmail.com </EMAIL_ADDRESS>
      <TELEPHONE_NUMBER> 2145134119 </TELEPHONE_NUMBER>
      <SSN> 295-13-4119 </SSN>
      <BIRTH_DAY> 1970-01-01 </BIRTH_DAY>
      <DEFAULT_SHIP_METHOD> AIR </DEFAULT_SHIP_METHOD>
      <EMAIL_NOTIFICATION> 1 </EMAIL_NOTIFICATION>
      <NEWS_LETTTER> 0 </NEWS_LETTTER>
      <ONLINE_STATEMENT> 1 </ONLINE_STATEMENT>
    </ns0:CUSTOMER>
  - <ns0:CUSTOMER >
      <CUSTOMER_ID> CUSTOMER5 </CUSTOMER_ID>
      <FIRST_NAME> Michael </FIRST_NAME>
      <LAST_NAME> Snow </LAST_NAME>
      <CUSTOMER_SINCE> 2001-10-01 </CUSTOMER_SINCE>
```

8. Open `CustomerInfo.ds` in Source View to view the generated XQuery. The query should be similar to that displayed in Figure 18-14.

   Note: The automatic namespace assignments may not match.

Figure 18-14 Date Function Source View



```
declare namespace ns1="ld:DataServices/CustomerDB/CUSTOMER";
import schema namespace ns0="ld:DataServices/MyQueries/CUSTOMER" at "ld:DataServices/MyQuerie
declare namespace tns="ld:DataServices/MyQueries/CustomerInfo";

...
declare function tns:getAllCustomers() as element(ns0:CUSTOMER)* ...;

...
declare function tns:getCustomerByName($FirstName as xs:string, $LastName as xs:string) as el

...
declare function tns:getCustomerBySSN($SSN as xs:string) as element(ns0:CUSTOMER)* ...;

...
declare function tns:getCustomerByBirthYear($BirthYear as xs:integer) as element(ns0:CUSTOMER
    for $CUSTOMER in ns2:CUSTOMER()
    where fn:year-from-date($CUSTOMER/BIRTH_DAY) eq $BirthYear

    return
    <ns0:CUSTOMER>
        <CUSTOMER_ID>{fn:data($CUSTOMER/CUSTOMER_ID)}</CUSTOMER_ID>
        <FIRST_NAME>{fn:data($CUSTOMER/FIRST_NAME)}</FIRST_NAME>
        <LAST_NAME>{fn:data($CUSTOMER/LAST_NAME)}</LAST_NAME>
        <CUSTOMER_SINCE>{fn:data($CUSTOMER/CUSTOMER_SINCE)}</CUSTOMER_SINCE>
        <EMAIL_ADDRESS>{fn:data($CUSTOMER/EMAIL_ADDRESS)}</EMAIL_ADDRESS>
        <TELEPHONE_NUMBER>{fn:data($CUSTOMER/TELEPHONE_NUMBER)}</TELEPHONE_NUMBER>
        <SSN?>{fn:data($CUSTOMER/SSN)}</SSN>
        <BIRTH_DAY?>{fn:data($CUSTOMER/BIRTH_DAY)}</BIRTH_DAY>
        <DEFAULT_SHIP_METHOD?>{fn:data($CUSTOMER/DEFAULT_SHIP_METHOD)}</DEFAULT_SHIP_METHOD>
        <EMAIL_NOTIFICATION?>{fn:data($CUSTOMER/EMAIL_NOTIFICATION)}</EMAIL_NOTIFICATION>
        <NEWS_LETTTER?>{fn:data($CUSTOMER/NEWS_LETTTER)}</NEWS_LETTTER>
        <ONLINE_STATEMENT?>{fn:data($CUSTOMER/ONLINE_STATEMENT)}</ONLINE_STATEMENT>
    </ns0:CUSTOMER>

};
```

18.6: Creating Outer Joins and Order By Expressions

Outer joins return all records from one table even it doesn't contain values that match those in the other table. For example, an outer join of customers and orders reports all customers—even those without orders.

Objectives

In this exercise, you will:

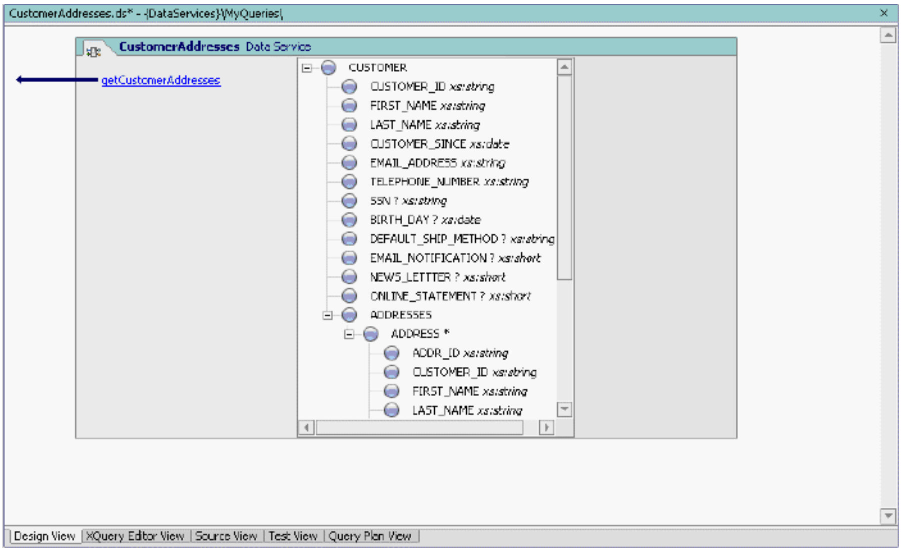
- Create a function that:
  - Returns customer information and their addresses (there may be more than 1).
  - Nests address information inside customer information.

  - Orders customers by first name and last name, in ascending order.
  - Orders addresses by zip code, in descending order.
- Test the function.
- Review the XQuery source code.

Instructions

1. Add a new data service to the MyQueries folder and name it CustomerAddresses.
2. Associate the `CustomerAddresses()` data service with the CUSTOMERADDRESS.xsd schema. The schema is located in `MyQueries\schemas`.
3. Add a new function to the CustomerAddresses data service and name it getCustomerAddresses.

Figure 18-15 Design View of CustomerAddresses Data Service

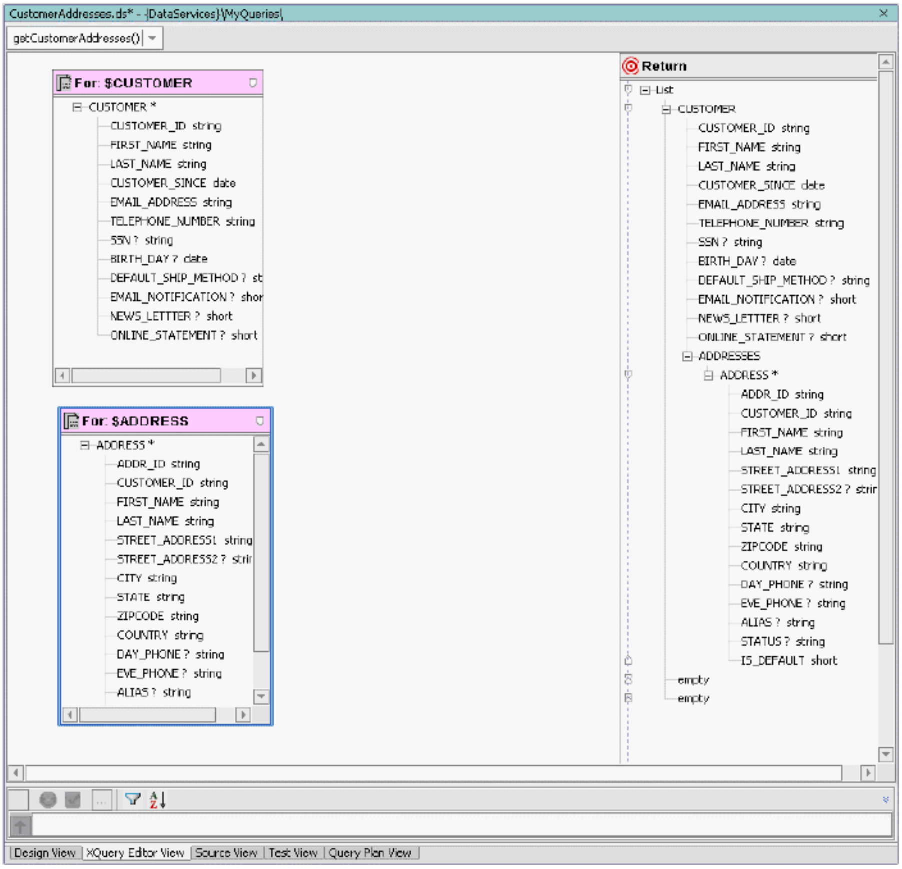


4. Click `getCustomerAddresses()` to open XQuery Editor View for that function.
5. Add two for nodes to the work area, by completing the following steps:
   a. In the Data Services Palette, expand DataServices\CustomerDB.

b. Open the `CUSTOMER.ds` folder (located in the CustomerDB folder), and then drag and drop `CUSTOMER()` into XQuery Editor View.

c. Open the `ADDRESS.ds` folder (located in the CustomerDB folder), and then drag and drop `ADDRESS()` into XQuery Editor View.

Figure 18-16 Source Nodes



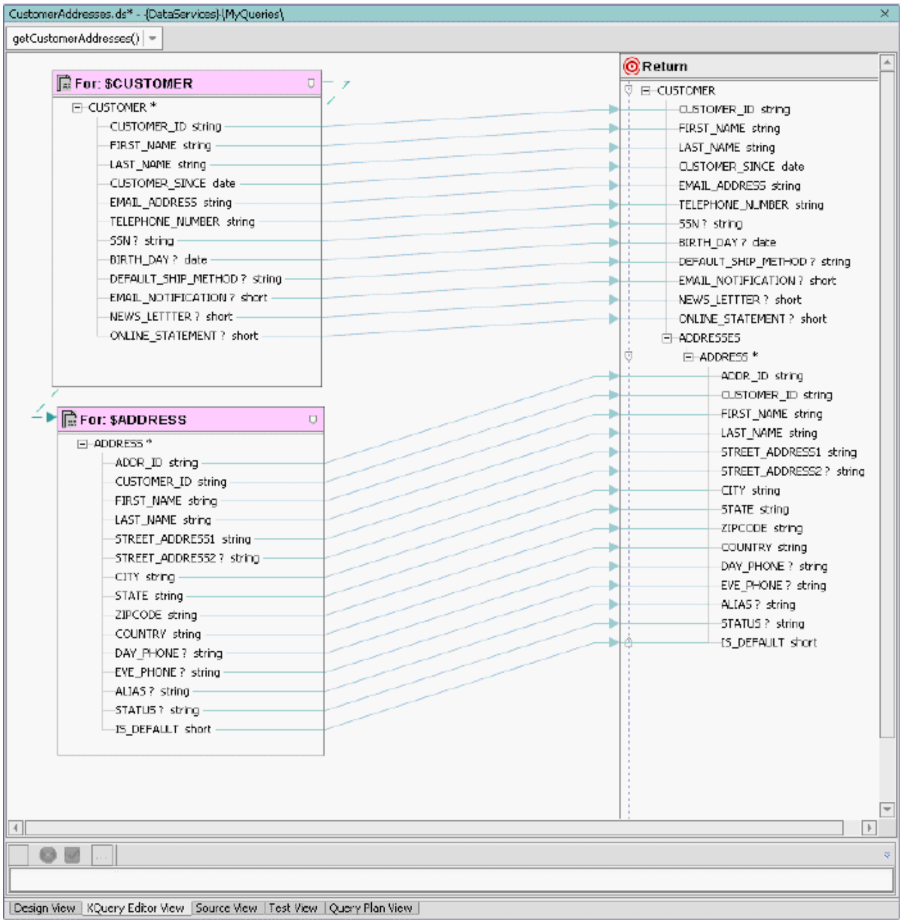
6. Create an induced mapping for the CUSTOMER node. Drag and drop the CUSTOMER* element (source) onto the CUSTOMER element (Return).

7. Create an induced mapping for the ADDRESS node. Drag and drop the ADDRESS* element (source) onto the ADDRESS element (Return).

Note: Do not drop the source element onto the ADDRESSES element.

8. Create a source node relationship. Drag and drop the CUSTOMER_ID element in the $CUSTOMER node onto the corresponding element in the $ADDRESS node.

Figure 18-17 Mapped and Joined Source Nodes



9. Add an OrderBy clause, by completing the following steps:

   a. Click the ADDRESS node's header. The Expression Editor opens.

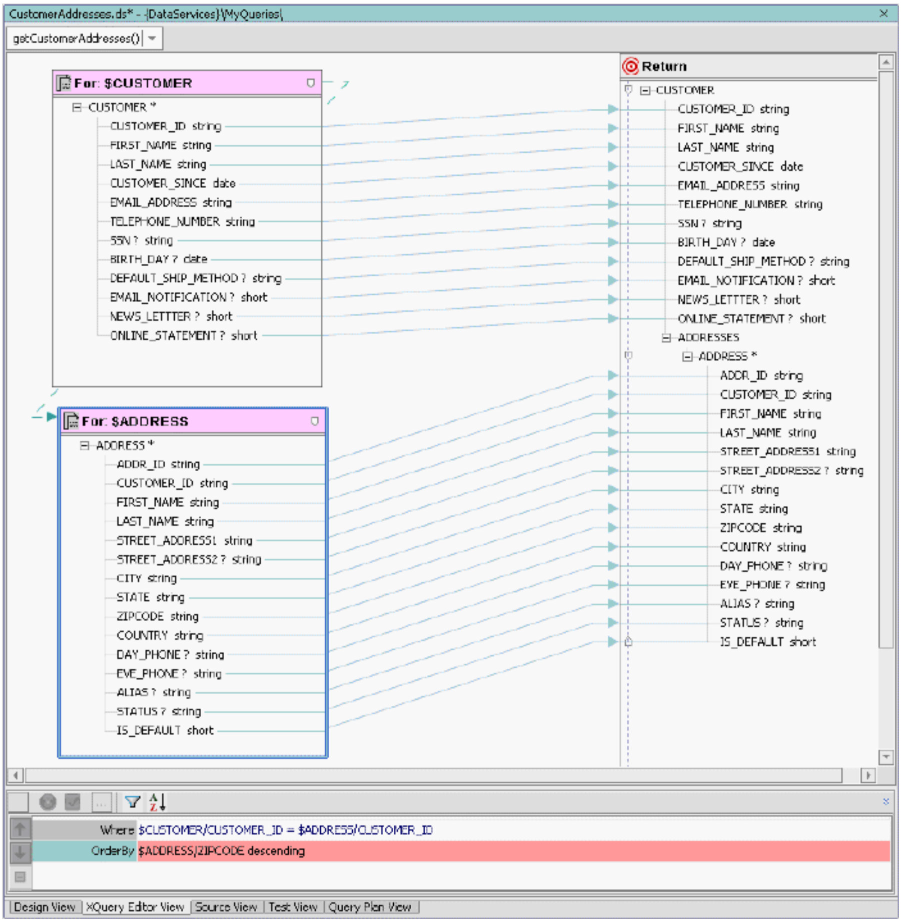
   b. Click the Order By Clause icon.

c. Click inside the Order By Clause field.

d. Enter $ADDRESS/ZIPCODE descending in the field.

e. Click the green check button to accept the changes.

Figure 18-18 OrderBy Clause



Where $CUSTOMER/CUSTOMER_ID = $ADDRESS/CUSTOMER_ID
OrderBy $ADDRESS/ZIPCODE descending

10. Test the function, by completing the following steps:

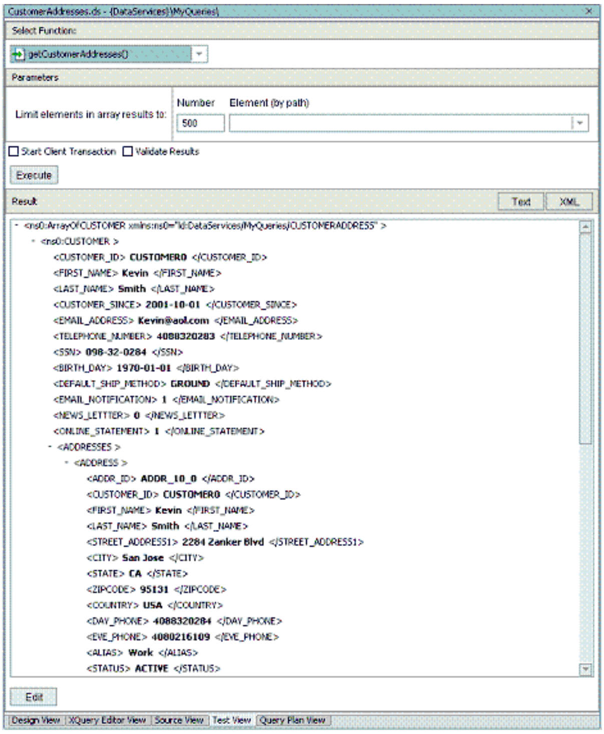
a. Open `CustomerAddresses.ds` in Test View.

b. Select `getCustomerAddresses()` from the function drop-down list.

c. Click Execute.

d. Confirm the results. Addresses should be nested after the customer's information.

Figure 18-19 Order By Test Results

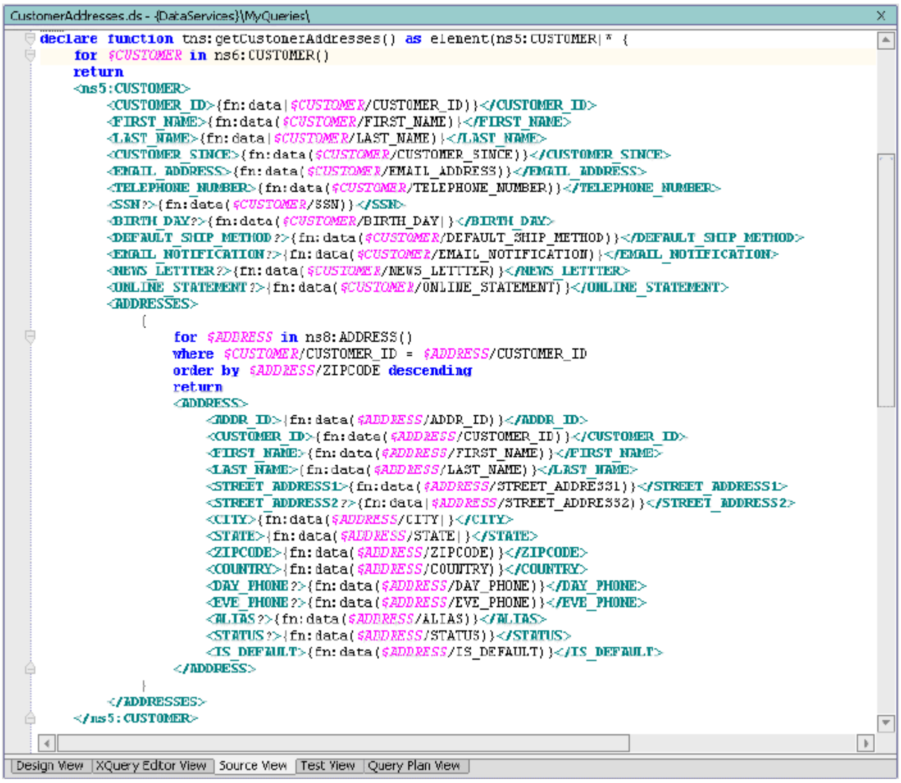


```
- <ns0:ArrayOfCUSTOMER xmlns:ns0="ld:DataServices/MyQueries/CUSTOMERADDRESS" >
  - <ns0:CUSTOMER >
      <CUSTOMER_ID> CUSTOMER0 </CUSTOMER_ID>
      <FIRST_NAME> Kevin </FIRST_NAME>
      <LAST_NAME> Smith </LAST_NAME>
      <CUSTOMER_SINCE> 2001-10-01 </CUSTOMER_SINCE>
      <EMAIL_ADDRESS> Kevin@aol.com </EMAIL_ADDRESS>
      <TELEPHONE_NUMBER> 4088320283 </TELEPHONE_NUMBER>
      <SSN> 098-32-0284 </SSN>
      <BIRTH_DAY> 1970-01-01 </BIRTH_DAY>
      <DEFAULT_SHIP_METHOD> GROUND </DEFAULT_SHIP_METHOD>
      <EMAIL_NOTIFICATION> 1 </EMAIL_NOTIFICATION>
      <NEWS_LETTTER> 0 </NEWS_LETTTER>
      <ONLINE_STATEMENT> 1 </ONLINE_STATEMENT>
    - <ADDRESSES >
      - <ADDRESS >
          <ADDR_ID> ADDR_10_0 </ADDR_ID>
          <CUSTOMER_ID> CUSTOMER0 </CUSTOMER_ID>
          <FIRST_NAME> Kevin </FIRST_NAME>
          <LAST_NAME> Smith </LAST_NAME>
          <STREET_ADDRESS1> 2284 Zanker Blvd </STREET_ADDRESS1>
          <CITY> San Jose </CITY>
          <STATE> CA </STATE>
          <ZIPCODE> 95131 </ZIPCODE>
          <COUNTRY> USA </COUNTRY>
          <DAY_PHONE> 4088320284 </DAY_PHONE>
          <EVE_PHONE> 4080216109 </EVE_PHONE>
          <ALIAS> Work </ALIAS>
          <STATUS> ACTIVE </STATUS>
```

11. Open `CustomerAddresses.ds` in Source View to view the generated XQuery.

    Note: The automatic namespace assignments may not match.

Figure 18-20 CustomerAddresses() Source View



```
declare function tns:getCustomerAddresses() as element(ns5:CUSTOMER)* {
    for $CUSTOMER in ns6:CUSTOMER()
    return
    <ns5:CUSTOMER>
        <CUSTOMER_ID>{fn:data($CUSTOMER/CUSTOMER_ID)}</CUSTOMER_ID>
        <FIRST_NAME>{fn:data($CUSTOMER/FIRST_NAME)}</FIRST_NAME>
        <LAST_NAME>{fn:data($CUSTOMER/LAST_NAME)}</LAST_NAME>
        <CUSTOMER_SINCE>{fn:data($CUSTOMER/CUSTOMER_SINCE)}</CUSTOMER_SINCE>
        <EMAIL_ADDRESS>{fn:data($CUSTOMER/EMAIL_ADDRESS)}</EMAIL_ADDRESS>
        <TELEPHONE_NUMBER>{fn:data($CUSTOMER/TELEPHONE_NUMBER)}</TELEPHONE_NUMBER>
        <SSN?>{fn:data($CUSTOMER/SSN)}</SSN>
        <BIRTH_DAY?>{fn:data($CUSTOMER/BIRTH_DAY)}</BIRTH_DAY>
        <DEFAULT_SHIP_METHOD?>{fn:data($CUSTOMER/DEFAULT_SHIP_METHOD)}</DEFAULT_SHIP_METHOD>
        <EMAIL_NOTIFICATION?>{fn:data($CUSTOMER/EMAIL_NOTIFICATION)}</EMAIL_NOTIFICATION>
        <NEWS_LETTTER?>{fn:data($CUSTOMER/NEWS_LETTTER)}</NEWS_LETTTER>
        <ONLINE_STATEMENT?>{fn:data($CUSTOMER/ONLINE_STATEMENT)}</ONLINE_STATEMENT>
        <ADDRESSES>
            {
                for $ADDRESS in ns8:ADDRESS()
                where $CUSTOMER/CUSTOMER_ID = $ADDRESS/CUSTOMER_ID
                order by $ADDRESS/ZIPCODE descending
                return
                <ADDRESS>
                    <ADDR_ID>{fn:data($ADDRESS/ADDR_ID)}</ADDR_ID>
                    <CUSTOMER_ID>{fn:data($ADDRESS/CUSTOMER_ID)}</CUSTOMER_ID>
                    <FIRST_NAME>{fn:data($ADDRESS/FIRST_NAME)}</FIRST_NAME>
                    <LAST_NAME>{fn:data($ADDRESS/LAST_NAME)}</LAST_NAME>
                    <STREET_ADDRESS1>{fn:data($ADDRESS/STREET_ADDRESS1)}</STREET_ADDRESS1>
                    <STREET_ADDRESS2?>{fn:data($ADDRESS/STREET_ADDRESS2)}</STREET_ADDRESS2>
                    <CITY>{fn:data($ADDRESS/CITY)}</CITY>
                    <STATE>{fn:data($ADDRESS/STATE)}</STATE>
                    <ZIPCODE>{fn:data($ADDRESS/ZIPCODE)}</ZIPCODE>
                    <COUNTRY>{fn:data($ADDRESS/COUNTRY)}</COUNTRY>
                    <DAY_PHONE?>{fn:data($ADDRESS/DAY_PHONE)}</DAY_PHONE>
                    <EVE_PHONE?>{fn:data($ADDRESS/EVE_PHONE)}</EVE_PHONE>
                    <ALIAS?>{fn:data($ADDRESS/ALIAS)}</ALIAS>
                    <STATUS?>{fn:data($ADDRESS/STATUS)}</STATUS>
                    <IS_DEFAULT>{fn:data($ADDRESS/IS_DEFAULT)}</IS_DEFAULT>
                </ADDRESS>
            }
        </ADDRESSES>
    </ns5:CUSTOMER>
```

18.7: Creating Group By and Aggregate Expressions

Sometimes, you may want to group data according to particular data elements, such as grouping customers by state and country.

Objectives

In this exercise, you will:

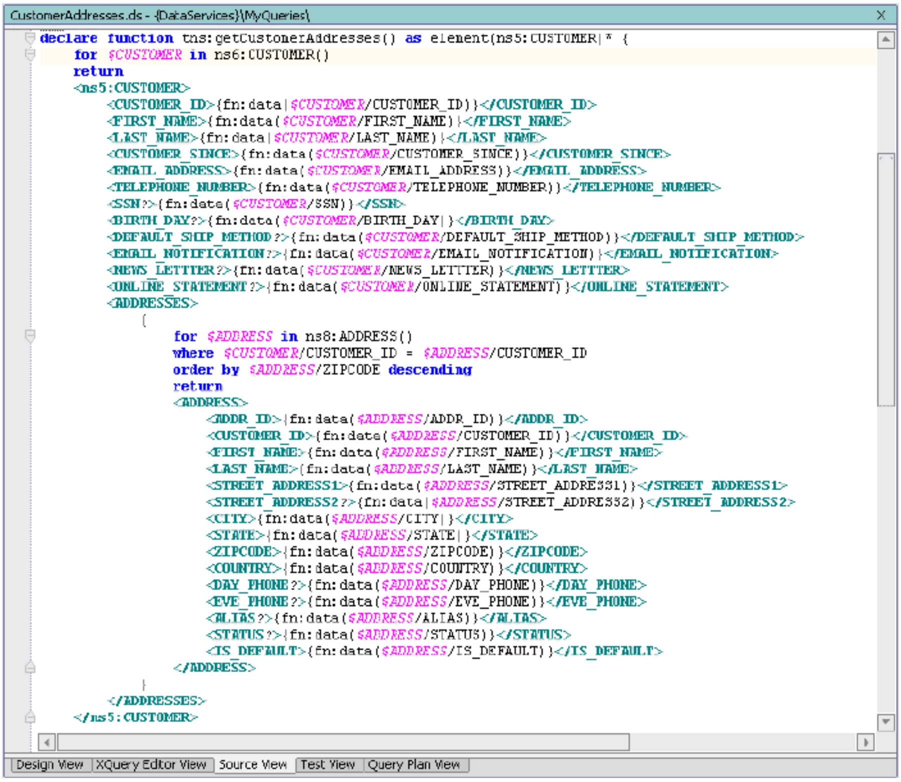
- Create a query using the group by operator and `sum()` function that generates a report of customers grouped by state and city, showing total sales by city.
- Test the function.

- Review the XQuery source code.

Instructions

1. Create a new data service in the MyQueries folder and name it CustomerOrders.
2. Associate the CustomerOrders data service with the `CUSTOMER_ORDER.xsd` schema. The schema is located in `MyQueries\schemas`.
3. Create a new function and name it getCustomerOrderAmount.

Figure 18-21 Design View of Customer Orders Data Service



```
declare function tns:getCustomerAddresses() as element(ns5:CUSTOMER)* {
    for $CUSTOMER in ns6:CUSTOMER()
    return
    <ns5:CUSTOMER>
        <CUSTOMER_ID>{fn:data($CUSTOMER/CUSTOMER_ID)}</CUSTOMER_ID>
        <FIRST_NAME>{fn:data($CUSTOMER/FIRST_NAME)}</FIRST_NAME>
        <LAST_NAME>{fn:data($CUSTOMER/LAST_NAME)}</LAST_NAME>
        <CUSTOMER_SINCE>{fn:data($CUSTOMER/CUSTOMER_SINCE)}</CUSTOMER_SINCE>
        <EMAIL_ADDRESS>{fn:data($CUSTOMER/EMAIL_ADDRESS)}</EMAIL_ADDRESS>
        <TELEPHONE_NUMBER>{fn:data($CUSTOMER/TELEPHONE_NUMBER)}</TELEPHONE_NUMBER>
        <SSN?>{fn:data($CUSTOMER/SSN)}</SSN>
        <BIRTH_DAY?>{fn:data($CUSTOMER/BIRTH_DAY)}</BIRTH_DAY>
        <DEFAULT_SHIP_METHOD?>{fn:data($CUSTOMER/DEFAULT_SHIP_METHOD)}</DEFAULT_SHIP_METHOD>
        <EMAIL_NOTIFICATION?>{fn:data($CUSTOMER/EMAIL_NOTIFICATION)}</EMAIL_NOTIFICATION>
        <NEWS_LETTTER?>{fn:data($CUSTOMER/NEWS_LETTTER)}</NEWS_LETTTER>
        <ONLINE_STATEMENT?>{fn:data($CUSTOMER/ONLINE_STATEMENT)}</ONLINE_STATEMENT>
        <ADDRESSES>
            {
                for $ADDRESS in ns8:ADDRESS()
                where $CUSTOMER/CUSTOMER_ID = $ADDRESS/CUSTOMER_ID
                order by $ADDRESS/ZIPCODE descending
                return
                <ADDRESS>
                    <ADDR_ID>{fn:data($ADDRESS/ADDR_ID)}</ADDR_ID>
                    <CUSTOMER_ID>{fn:data($ADDRESS/CUSTOMER_ID)}</CUSTOMER_ID>
                    <FIRST_NAME>{fn:data($ADDRESS/FIRST_NAME)}</FIRST_NAME>
                    <LAST_NAME>{fn:data($ADDRESS/LAST_NAME)}</LAST_NAME>
                    <STREET_ADDRESS1>{fn:data($ADDRESS/STREET_ADDRESS1)}</STREET_ADDRESS1>
                    <STREET_ADDRESS2?>{fn:data($ADDRESS/STREET_ADDRESS2)}</STREET_ADDRESS2>
                    <CITY>{fn:data($ADDRESS/CITY)}</CITY>
                    <STATE>{fn:data($ADDRESS/STATE)}</STATE>
                    <ZIPCODE>{fn:data($ADDRESS/ZIPCODE)}</ZIPCODE>
                    <COUNTRY>{fn:data($ADDRESS/COUNTRY)}</COUNTRY>
                    <DAY_PHONE?>{fn:data($ADDRESS/DAY_PHONE)}</DAY_PHONE>
                    <EVE_PHONE?>{fn:data($ADDRESS/EVE_PHONE)}</EVE_PHONE>
                    <ALIAS?>{fn:data($ADDRESS/ALIAS)}</ALIAS>
                    <STATUS?>{fn:data($ADDRESS/STATUS)}</STATUS>
                    <IS_DEFAULT>{fn:data($ADDRESS/IS_DEFAULT)}</IS_DEFAULT>
                </ADDRESS>
            }
        </ADDRESSES>
    </ns5:CUSTOMER>
```

4. Click getCustomerOrderAmount to open XQuery Editor View for that function.
5. Add a for node, by completing the following steps:

a. In the Data Services Palette, open the `CUSTOMER_ORDER.ds` folder, which is located in `DataServices\ApparelDB`.

b. Drag and drop `CUSTOMER_ORDER()` into XQuery Editor View.

6. Create a GroupBy clause, by completing the following steps:

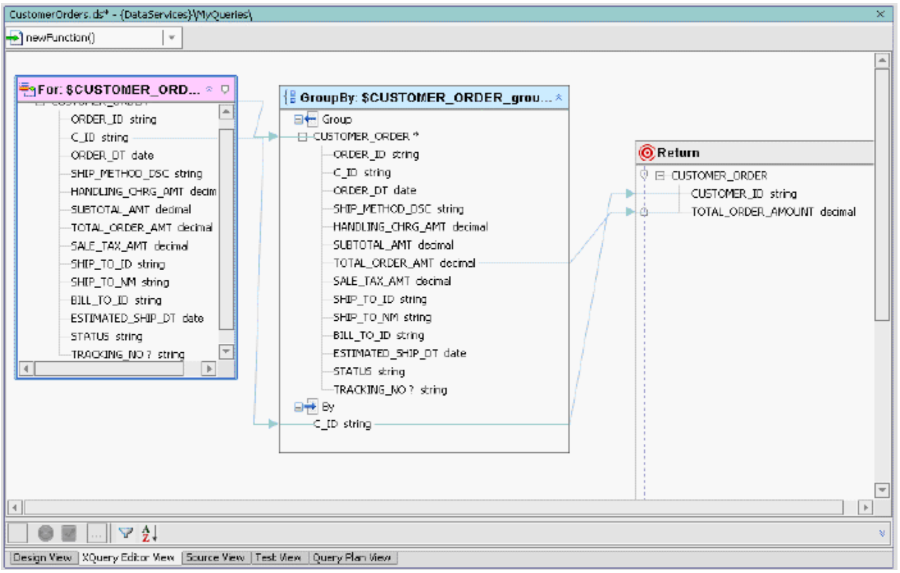
a. Right-click the C_ID element in the $CUSTOMER_ORDER source node.

b. Choose Create Group By. A GroupBy node is created.

7. Create a simple mapping. Drag and drop the TOTAL_ORDER_AMT from the Group section of the GroupBy node onto the corresponding element in the Return type.

8. Create a simple mapping. Drag and drop the C_ID element in the By section of the GroupBy node to the corresponding element in the Return type.

Figure 18-22 GroupBy Node Added and Mapped



Modify a Return expression, by completing the following steps:

a. Click the TOTAL_ORDER_AMOUNT, located in the Return node. The Expression Editor opens. Every element in a Return type has an underlying expression. In this case the expression is:

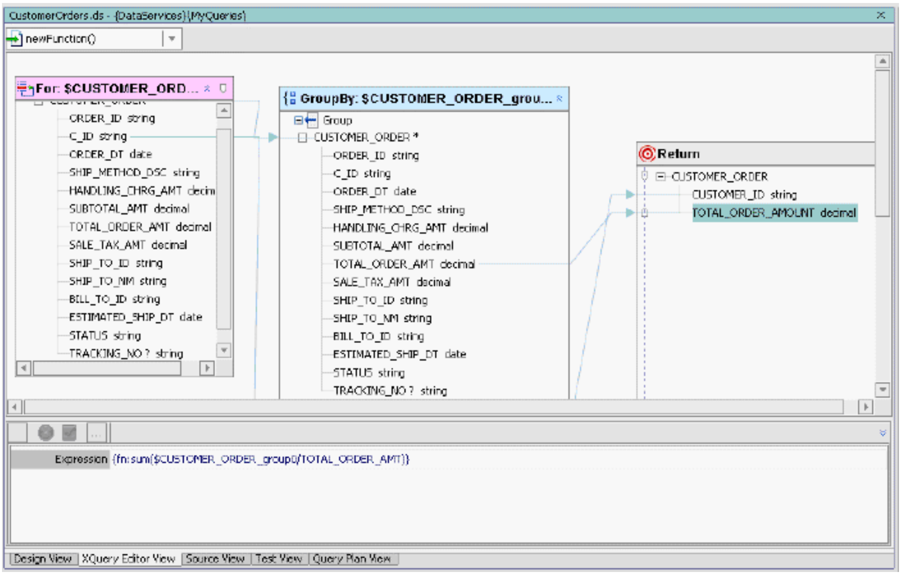
```
{fn:data($CUSTOMER_ORDER_group/TOTAL_ORDER_AMT)}
```

b. Edit the expression so that it changes `fn:data()` to `fn:sum()`, as follows:

```
{fn:sum($CUSTOMER_ORDER_group/TOTAL_ORDER_AMT)}
```

c. Click the green check button to accept the changes.

Figure 18-23 Aggregate Expression



Expression {fn:sum($CUSTOMER_ORDER_group0/TOTAL_ORDER_AMT)}

9. Test the function, by completing the following steps:

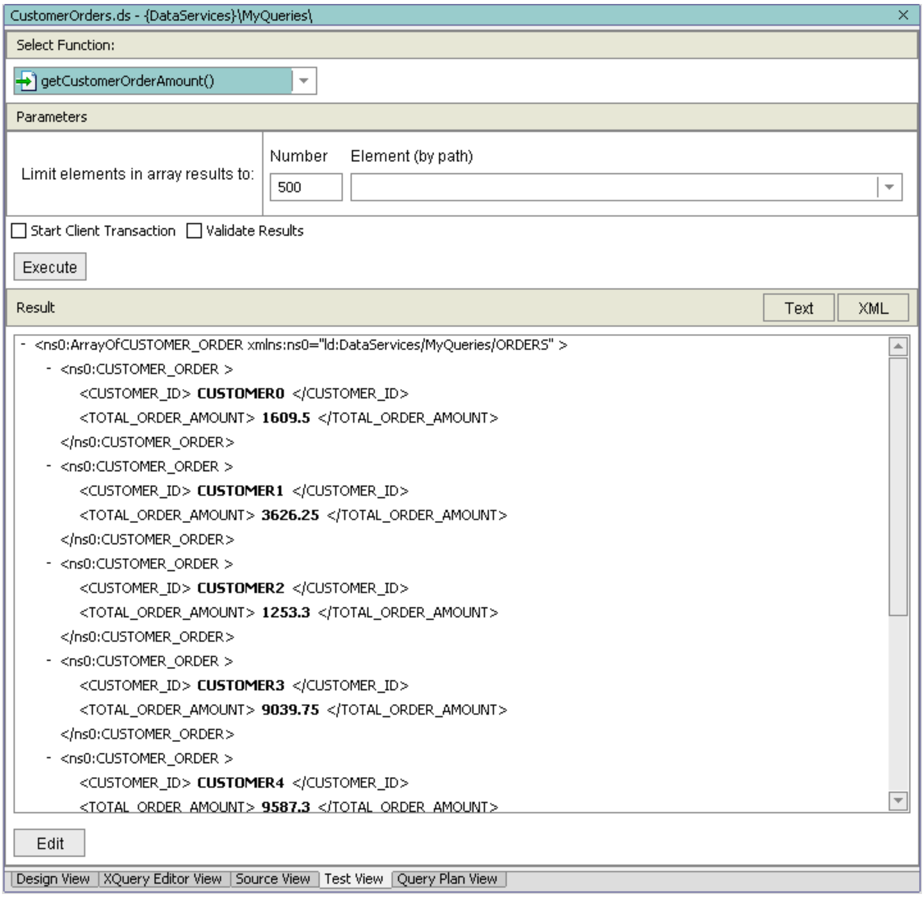
   a. Open `CustomerOrders.ds` in Test View.

   b. Select `getCustomerOrderAmount()` from the Function drop-down list.

   c. Click Execute.

   d. Confirm the results.

Figure 18-24 Aggregate Test Results

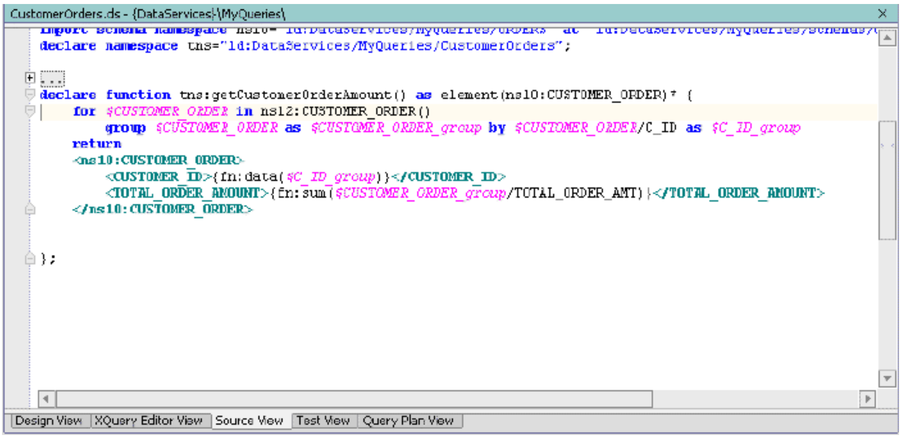


```
- <ns0:ArrayOfCUSTOMER_ORDER xmlns:ns0="ld:DataServices/MyQueries/ORDERS" >
  - <ns0:CUSTOMER_ORDER >
      <CUSTOMER_ID> CUSTOMER0 </CUSTOMER_ID>
      <TOTAL_ORDER_AMOUNT> 1609.5 </TOTAL_ORDER_AMOUNT>
    </ns0:CUSTOMER_ORDER>
  - <ns0:CUSTOMER_ORDER >
      <CUSTOMER_ID> CUSTOMER1 </CUSTOMER_ID>
      <TOTAL_ORDER_AMOUNT> 3626.25 </TOTAL_ORDER_AMOUNT>
    </ns0:CUSTOMER_ORDER>
  - <ns0:CUSTOMER_ORDER >
      <CUSTOMER_ID> CUSTOMER2 </CUSTOMER_ID>
      <TOTAL_ORDER_AMOUNT> 1253.3 </TOTAL_ORDER_AMOUNT>
    </ns0:CUSTOMER_ORDER>
  - <ns0:CUSTOMER_ORDER >
      <CUSTOMER_ID> CUSTOMER3 </CUSTOMER_ID>
      <TOTAL_ORDER_AMOUNT> 9039.75 </TOTAL_ORDER_AMOUNT>
    </ns0:CUSTOMER_ORDER>
  - <ns0:CUSTOMER_ORDER >
      <CUSTOMER_ID> CUSTOMER4 </CUSTOMER_ID>
      <TOTAL_ORDER_AMOUNT> 9587.3 </TOTAL_ORDER_AMOUNT>
```

10. Open `CustomerOrders.ds` in Source View to view the generated XQuery.

    Note: The automatic namespace assignments may not match that shown in the exercise.

Figure 18-25 Source View of the CustomerOrders Data Service



```
import schema namespace ns10="ld:DataServices/MyQueries/ORDERS" at "ld:DataServices/MyQueries/schemas/
declare namespace tns="ld:DataServices/MyQueries/CustomerOrders";

...
declare function tns:getCustomerOrderAmount() as element(ns10:CUSTOMER_ORDER)* {
    for $CUSTOMER_ORDER in ns12:CUSTOMER_ORDER()
        group $CUSTOMER_ORDER as $CUSTOMER_ORDER_group by $CUSTOMER_ORDER/C_ID as $C_ID_group
    return
    <ns10:CUSTOMER_ORDER>
        <CUSTOMER_ID>{fn:data($C_ID_group)}</CUSTOMER_ID>
        <TOTAL_ORDER_AMOUNT>{fn:sum($CUSTOMER_ORDER_group/TOTAL_ORDER_AMT)}</TOTAL_ORDER_AMOUNT>
    </ns10:CUSTOMER_ORDER>


};
```

18.8: Creating Constant Expressions

Creating a data service query that uses a constant expression enables a quick and easy way to locate specific information. For example, you can use a constant expression to identify all customers who ship by Ground method.

Objectives

In this exercise, you will:

- Create a non-parameterized function that will return all customers whose default shipping method is GROUND.
- Test the function.
- View the XQuery source code.

Instructions

1. Add a new function to the CustomerInfo data service and name it getGroundCustomers.
2. Click the `getGroundCustomers()` function to open the XQuery Editor View.
3. Add a for node, by completing the following steps:

a. In the Data Services Palette, open the `CUSTOMER.ds` folder, which is located in the `DataServices\CustomerDB` folder.

b. Drag and drop `CUSTOMER()` into XQuery Editor View.

4. Create an induced mapping. Drag and drop the entire CUSTOMER* element (source node) onto the CUSTOMER element (Return).

5. Add a where clause, by completing the following steps:

a. Click the CUSTOMER node's header. The Expression Editor opens.

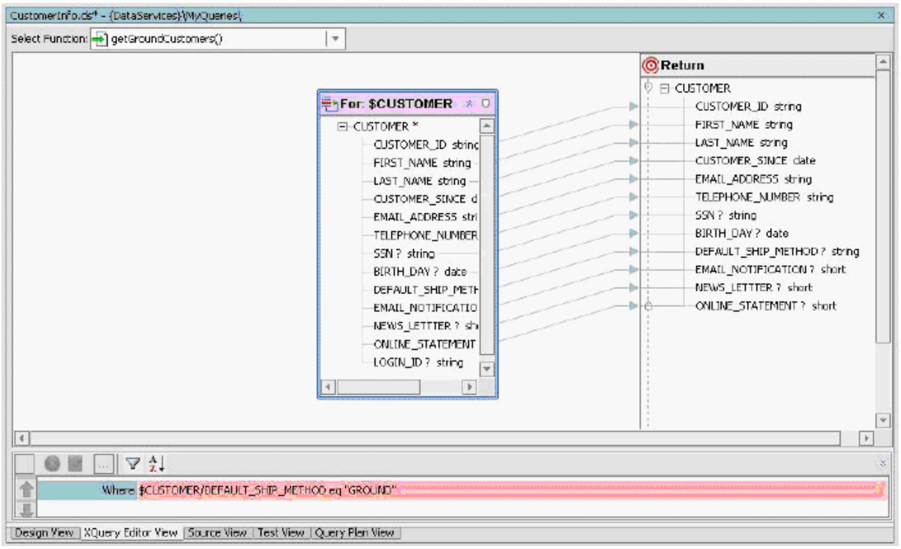
b. Click the Add Where Clause icon.

c. Enter the following expression as a where clause:

```
$CUSTOMER/DEFAULT_SHIP_METHOD eq "GROUND"
```

d. Click the green check mark icon to accept the where clause for the customer object.

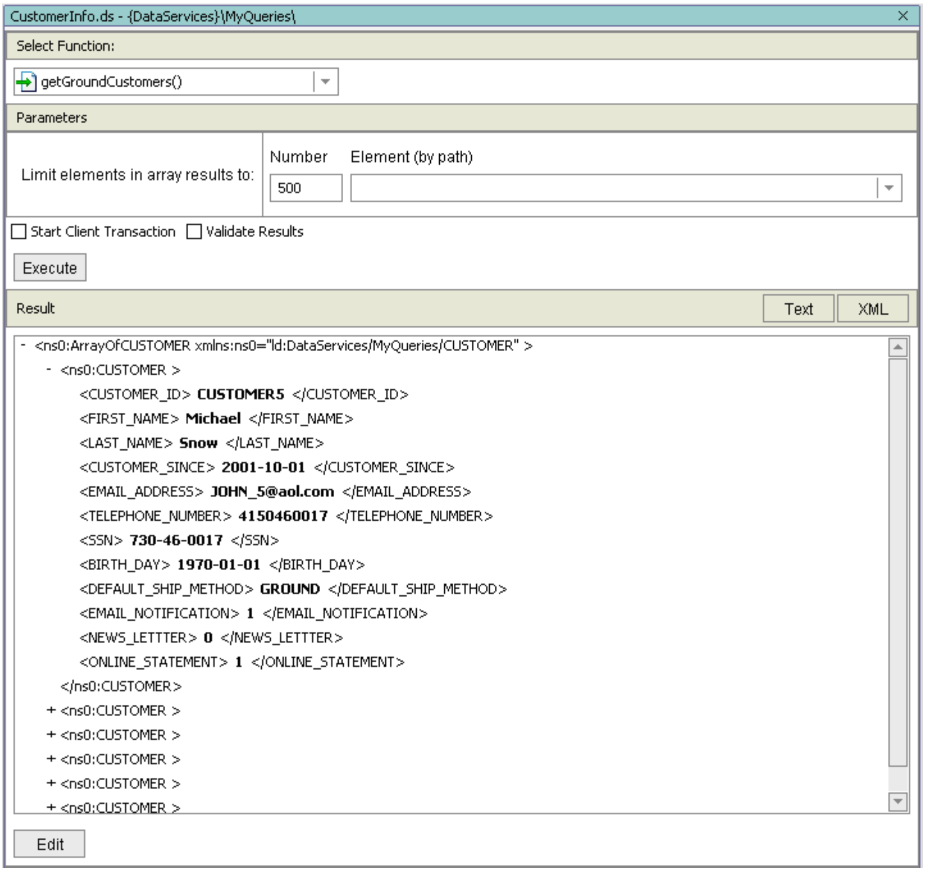
Figure 18-26 Constant Function with Default Expression



```
Where $CUSTOMER/DEFAULT_SHIP_METHOD eq "GROUND"
```

6. Test the function. The results should be as displayed in Figure 18-27.

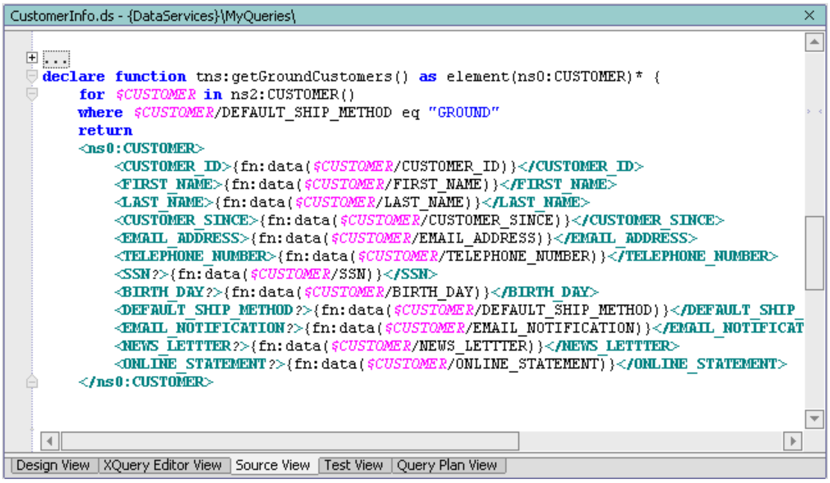
Figure 18-27 Test Results of a Constant Expression



```
- <ns0:ArrayOfCUSTOMER xmlns:ns0="ld:DataServices/MyQueries/CUSTOMER" >
  - <ns0:CUSTOMER >
      <CUSTOMER_ID> CUSTOMER5 </CUSTOMER_ID>
      <FIRST_NAME> Michael </FIRST_NAME>
      <LAST_NAME> Snow </LAST_NAME>
      <CUSTOMER_SINCE> 2001-10-01 </CUSTOMER_SINCE>
      <EMAIL_ADDRESS> JOHN_5@aol.com </EMAIL_ADDRESS>
      <TELEPHONE_NUMBER> 4150460017 </TELEPHONE_NUMBER>
      <SSN> 730-46-0017 </SSN>
      <BIRTH_DAY> 1970-01-01 </BIRTH_DAY>
      <DEFAULT_SHIP_METHOD> GROUND </DEFAULT_SHIP_METHOD>
      <EMAIL_NOTIFICATION> 1 </EMAIL_NOTIFICATION>
      <NEWS_LETTTER> 0 </NEWS_LETTTER>
      <ONLINE_STATEMENT> 1 </ONLINE_STATEMENT>
    </ns0:CUSTOMER>
  + <ns0:CUSTOMER >
  + <ns0:CUSTOMER >
  + <ns0:CUSTOMER >
  + <ns0:CUSTOMER >
  + <ns0:CUSTOMER >
```

7. Open `CustomerInfo.ds` in Source View. The code should be as displayed in Figure 18-28.

Figure 18-28 Source Code



```
...
declare function tns:getGroundCustomers() as element(ns0:CUSTOMER)* {
    for $CUSTOMER in ns2:CUSTOMER()
    where $CUSTOMER/DEFAULT_SHIP_METHOD eq "GROUND"
    return
    <ns0:CUSTOMER>
        <CUSTOMER_ID>{fn:data($CUSTOMER/CUSTOMER_ID)}</CUSTOMER_ID>
        <FIRST_NAME>{fn:data($CUSTOMER/FIRST_NAME)}</FIRST_NAME>
        <LAST_NAME>{fn:data($CUSTOMER/LAST_NAME)}</LAST_NAME>
        <CUSTOMER_SINCE>{fn:data($CUSTOMER/CUSTOMER_SINCE)}</CUSTOMER_SINCE>
        <EMAIL_ADDRESS>{fn:data($CUSTOMER/EMAIL_ADDRESS)}</EMAIL_ADDRESS>
        <TELEPHONE_NUMBER>{fn:data($CUSTOMER/TELEPHONE_NUMBER)}</TELEPHONE_NUMBER>
        <SSN?>{fn:data($CUSTOMER/SSN)}</SSN>
        <BIRTH_DAY?>{fn:data($CUSTOMER/BIRTH_DAY)}</BIRTH_DAY>
        <DEFAULT_SHIP_METHOD?>{fn:data($CUSTOMER/DEFAULT_SHIP_METHOD)}</DEFAULT_SHIP_
        <EMAIL_NOTIFICATION?>{fn:data($CUSTOMER/EMAIL_NOTIFICATION)}</EMAIL_NOTIFICAT
        <NEWS_LETTTER?>{fn:data($CUSTOMER/NEWS_LETTTER)}</NEWS_LETTTER>
        <ONLINE_STATEMENT?>{fn:data($CUSTOMER/ONLINE_STATEMENT)}</ONLINE_STATEMENT>
    </ns0:CUSTOMER>
```

Lesson Summary

In this lesson you learned how to:

- Use the graphical XQuery Editor View to create parameterized, string, and date functions; outer joins, aggregate, and order by and constant expressions.

Use the XQuery Function Palette to add built-in XQuery functions to a query.

Building XQueries in Source View

In the previous lesson, you built XQueries using XQuery Editor View. Sometimes, it is necessary to programmatically build a query or modify its code. In this lesson, you will learn how to use Source View to create and edit query functions.

Objectives

After completing this lesson, you will be able to:

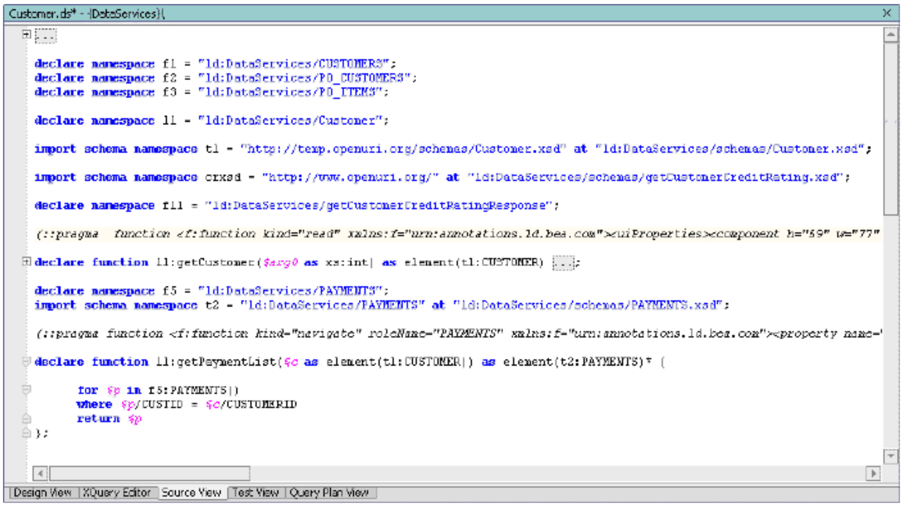
- Use Source View to add, edit, or delete XQuery code that defines a data service's query functions including creating:
  - A new XML type
  - A parameterized Xquery
  - Inner and outer joins
  - A multi-level group by
  - If-then-else if constructs
  - A union and concatenation operation
- Compare the coded query with the XQuery Editor View.

Overview

Source View lets you view and/or modify the data service's XQuery source code. In general, a data service is simply a file that contains XQuery code. Although ALDSP provides extensive visual design tools for developing a data service, sometimes you may need to work directly with XQuery syntax.

Two-way editing is supported—changes you make in Source View are reflected in XQuery Editor View, and vice versa. The source code is commented to help you edit the source correctly.

Figure 19-1 Source View Sample

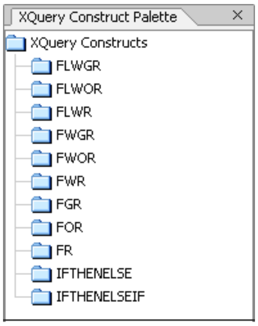


```
...

declare namespace f1 = "ld:DataServices/CUSTOMERS";
declare namespace f2 = "ld:DataServices/PO_CUSTOMERS";
declare namespace f3 = "ld:DataServices/PO_ITEMS";

declare namespace l1 = "ld:DataServices/Customer";

import schema namespace t1 = "http://temp.openuri.org/schemas/Customer.xsd" at "ld:DataServices/schemas/Customer.xsd";

import schema namespace crxsd = "http://www.openuri.org/" at "ld:DataServices/schemas/getCustomerCreditRating.xsd";

declare namespace f11 = "ld:DataServices/getCustomerCreditRatingResponse";

(::pragma  function <f:function kind="read" xmlns:f="urn:annotations.ld.bea.com"><uiProperties><component h="69" w="77"

declare function l1:getCustomer($arg0 as xs:int) as element(t1:CUSTOMER) ...;

declare namespace f5 = "ld:DataServices/PAYMENTS";
import schema namespace t2 = "ld:DataServices/PAYMENTS" at "ld:DataServices/schemas/PAYMENTS.xsd";

(::pragma function <f:function kind="navigate" roleName="PAYMENTS" xmlns:f="urn:annotations.ld.bea.com"><property name="

declare function l1:getPaymentList($c as element(t1:CUSTOMER)) as element(t2:PAYMENTS)* {

        for $p in f5:PAYMENTS()
        where $p/CUSTID = $c/CUSTOMERID
        return $p
};
```

Source View Tools

Within Source View, you can use the XQuery Construct Palette, which lets you add any of several built-in generic FLWOR statements to the XQuery syntax. You can then customize the generic statement to match your particular needs.

Figure 19-2 Query Construct Palette



To add a FLWOR construct, drag and drop the selected item into the appropriate declare function space.

If XQuery Construct Palette is not open, choose View → Windows → XQuery Construct Palette.

19.1 Creating a New XML Type

For each of the queries created in this lesson, you will define a function that returns results nested within the Return type. To enable that, you need to create a data service with an undefined XML type. By leaving the XML type's schema undefined, you can modify the Return type on an ad hoc basis, without a need to be concerned about synchronizing the XML and Return types.

Objectives

In this exercise, you will:

- Create a new data service, called `XQueries.ds`.
- Create a new, but undefined, XML type.

Instructions

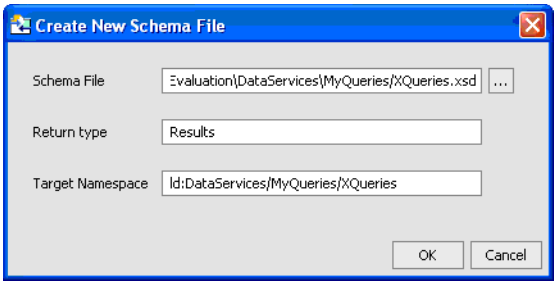
1. Create a new data service in the MyQueries folder and name it XQueries.
2. Create a new XML type by completing the following steps:
   a. Right-click the XQueries Data Service header.
   b. Select Create XML Type.

c. Enter Results in the Return Type field.

Note: Do not change the default settings for the Schema File and Target Namespace fields.

d. Click OK.

Figure 19-3 Create New XML Type



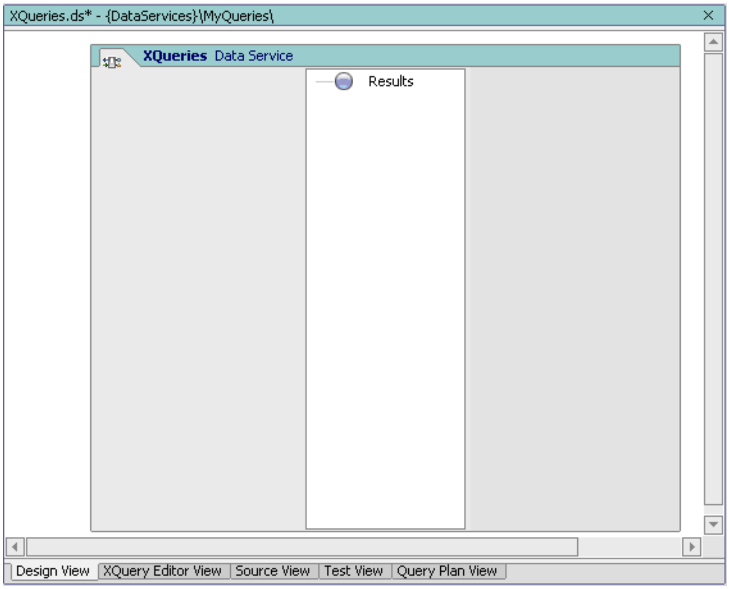
Schema File: Evaluation\DataServices\MyQueries/XQueries.xsd

Return type: Results

Target Namespace: ld:DataServices/MyQueries/XQueries

3. Confirm that the data service diagram is as displayed in Figure 19-4.

Figure 19-4 Design View: Undefined Results Type



19.2 Creating a Basic Parameterized XQuery

There are two basic types of queries: those without parameters and those with parameters. In the previous lesson, you used XQuery Editor View's graphical tools to define a query with parameters. In this exercise, you will use Source Editor to programmatically define a parameterized query.

Objectives

In this exercise, you will:

- Build a query that retrieves customer information based on first and last names.
- View the results in XQuery Editor View.
- Test the function.

Instructions

Note: Namespaces may differ for your application.

1. Add a new function to `XQueries.ds` and name it getCustomerByName.
2. Open Source View.
3. Define the function declaration, by completing the following steps:
   a. Add the following parameter to the first parenthesis:

      ```
      $p_firstname as xs:string, $p_lastname as xs:string
      ```

   b. Remove the asterisk (*), because you want this function to only return a single result.

      The code should be similar to the following:

      ```
      declare function tns:getCustomerByName($p_firstname as xs:string,
      $p_lastname as xs:string) as element(ns0:Results) {
      ```

4. Click the + symbol next to the `getCustomerByName()` function. This opens the function body.
5. Split the `<tns0:RESULTS/>` element into open and end tags, with curly braces in between for the XQuery. The code should be as follows (ignore the error indicator):

   ```
   <tns0:Results>

   {

   }
   ```

```
</tns0:Results>
```

6. Open XQuery Construct Palette.
7. Drag and drop the FWR construct between the curly braces. The code should be as follows:

   ```
   for $var in ()
   where true()
   return ()
   ```

8. Define the for clause by completing the following steps:
   a. Change the variable to $customer.
   b. In the Data Services Palette, expand `CustomerDB\CUSTOMER.ds`.
   c. Drag and drop `CUSTOMER()` into the for clause's first empty parenthesis. The code should be similar to the following:

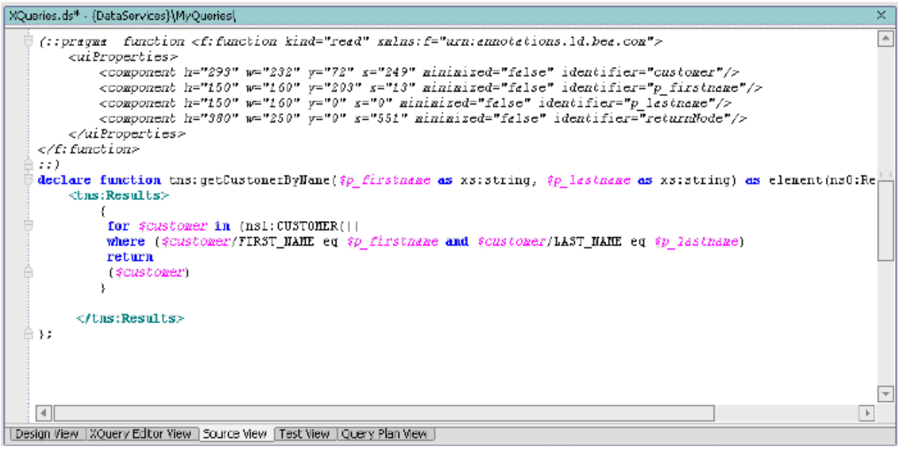
      ```
      for $customer in (ns1:CUSTOMER())
      where true ()
      return ()
      ```

9. Replace the where clause `true()` code with the following:

   ```
   $customer/FIRST_NAME eq $p_firstname and $customer/LAST_NAME eq
   $p_lastname
   ```

10. Set the return clause, by adding $customer between the parenthesis.
11. Confirm that the source code is as displayed in Figure 19-5; namespaces may be different for your application.

Figure 19-5 Parameterized Query Source Code

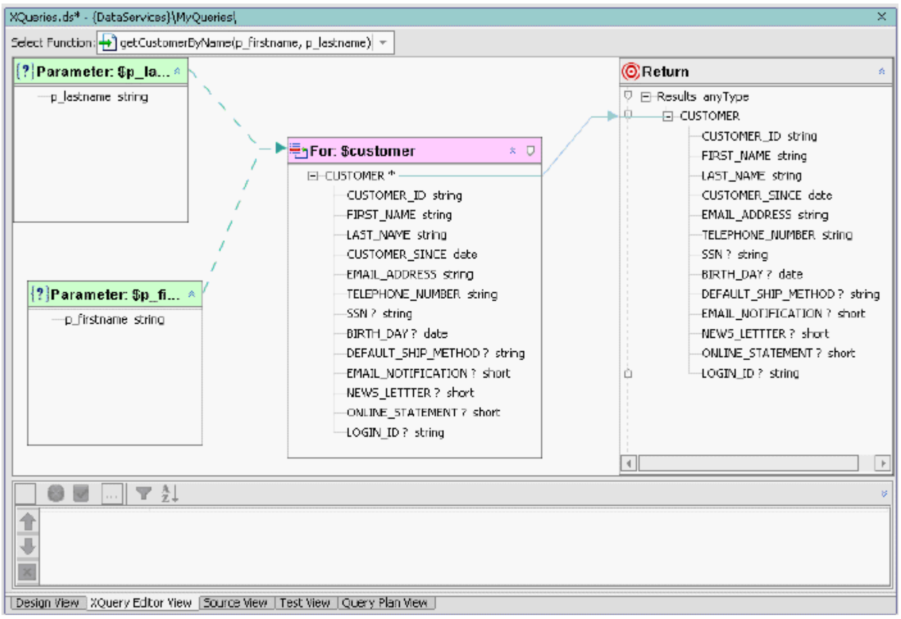


```
(::pragma  function <f:function kind="read" xmlns:f="urn:annotations.ld.bea.com">
    <uiProperties>
        <component h="293" w="232" y="72" x="249" minimized="false" identifier="customer"/>
        <component h="150" w="160" y="203" x="13" minimized="false" identifier="p_firstname"/>
        <component h="150" w="160" y="0" x="0" minimized="false" identifier="p_lastname"/>
        <component h="380" w="250" y="0" x="551" minimized="false" identifier="returnNode"/>
    </uiProperties>
</f:function>
::)
declare function tns:getCustomerByName($p_firstname as xs:string, $p_lastname as xs:string) as element(ns0:Re
    <tns:Results>
        {
         for $customer in (ns1:CUSTOMER()) 
         where ($customer/FIRST_NAME eq $p_firstname and $customer/LAST_NAME eq $p_lastname)
         return
         ($customer)
        }

     </tns:Results>
};
```

12. Build the DataServices project.

13. Open `XQueries.ds` in XQuery Editor View and review the graphical version of the XQuery code. It should be as displayed in Figure 19-6.

Figure 19-6 Query Editor View of Parameterized Function



14. Test the function, by completing the following steps:

    a. Open `XQueries.ds` in Test View.

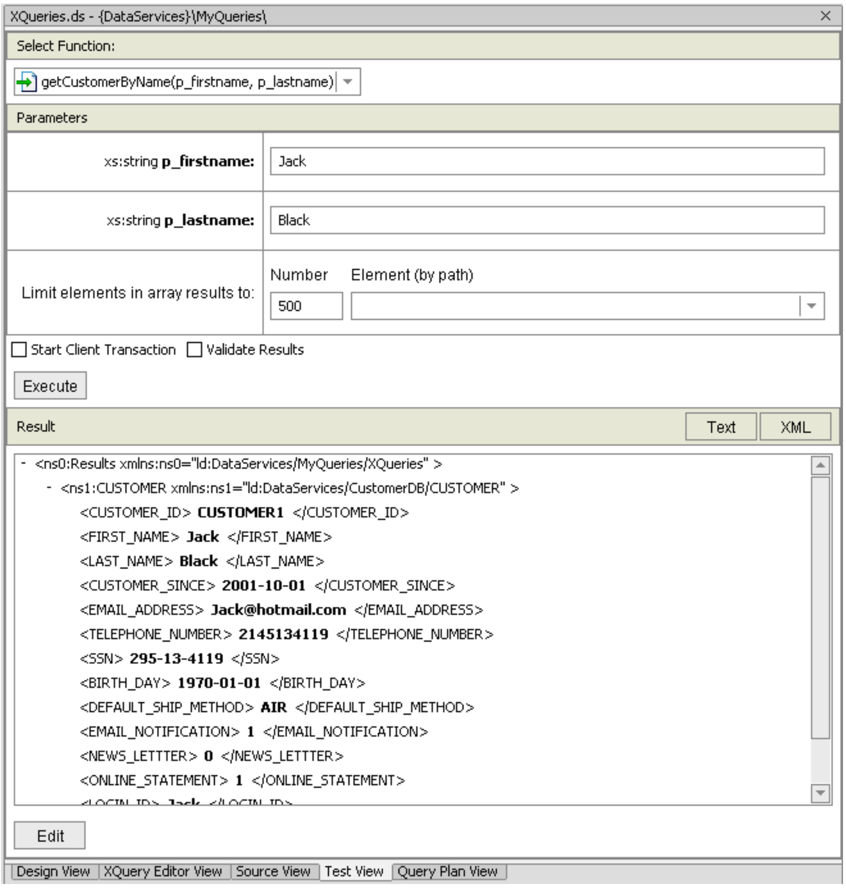
    b. Select `getCustomersByName()` from the Function drop-down list.

    c. Enter the following parameters:

    ```
    Firstname: Jack
    Lastname: Black
    ```

    d. Confirm the results.

Figure 19-7 Test Results of a Parameterized Function



getCustomerByName(p_firstname, p_lastname)

xs:string **p_firstname:** Jack

xs:string **p_lastname:** Black

Limit elements in array results to: Number 500 Element (by path)

```
- <ns0:Results xmlns:ns0="ld:DataServices/MyQueries/XQueries" >
   - <ns1:CUSTOMER xmlns:ns1="ld:DataServices/CustomerDB/CUSTOMER" >
        <CUSTOMER_ID> CUSTOMER1 </CUSTOMER_ID>
        <FIRST_NAME> Jack </FIRST_NAME>
        <LAST_NAME> Black </LAST_NAME>
        <CUSTOMER_SINCE> 2001-10-01 </CUSTOMER_SINCE>
        <EMAIL_ADDRESS> Jack@hotmail.com </EMAIL_ADDRESS>
        <TELEPHONE_NUMBER> 2145134119 </TELEPHONE_NUMBER>
        <SSN> 295-13-4119 </SSN>
        <BIRTH_DAY> 1970-01-01 </BIRTH_DAY>
        <DEFAULT_SHIP_METHOD> AIR </DEFAULT_SHIP_METHOD>
        <EMAIL_NOTIFICATION> 1 </EMAIL_NOTIFICATION>
        <NEWS_LETTTER> 0 </NEWS_LETTTER>
        <ONLINE_STATEMENT> 1 </ONLINE_STATEMENT>
```

15. (Optional) Open `CustomerInfo.ds` in XQuery Editor View and compare the diagrams for the two data services.

XQuery Code Reference for a Parameterized Function

```
declare function tns:getCustomerByName($p_firstname as xs:string,
$p_lastname as xs:string) as element(tns0:Results) {

   <tns:Results>

   {
         for $customer in (ns1:CUSTOMER())
        where ($customer/FIRST_NAME eq $p_firstname and $customer/LAST_NAME
         eq $p_lastname)
         return
             ($customer)
     }
     </tns:Results>
```

19.3 Creating a String Function

XQuery provides numerous string functions that can be incorporated into your business logic.

Objectives

In this exercise, you will:

- Create a `startwith()` function that retrieves customer information by name or SSN.
- Test the function.

Instructions

1. Add a new function to `XQueries.ds` and name it `getCustomerByNameorSSN()`.
2. Open `XQueries.ds` in Source View.
3. Define the function declaration, by changing the parameter as follows:

   ```
   $fullname as xs:string, $ssn as xs:string
   ```

4. Replace the contents of the where clause with the following:

   ```
   fn:contains(fn:upper-case(fn:concat($customer/FIRST_NAME,"
   ",$customer/LAST_NAME)), fn:upper-case($fullname) ) or
   fn:starts-with($customer/SSN, $ssn)
   ```

   Note: You can either type the code in or build the clause by using the following built-in functions, located in the XQuery Function Palette:

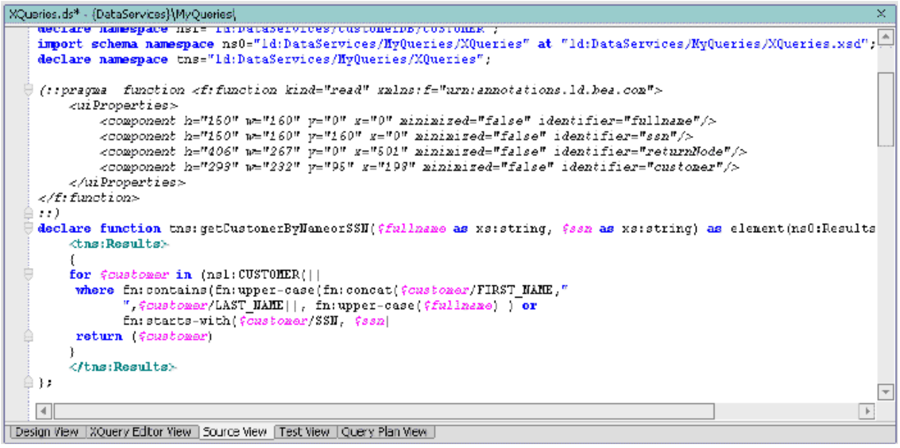
   ```
   fn:concat fn:starts-with
   fn:contains fn:upper-case
   ```

Note: The full name is created "on-the-spot" by concatenating FIRST_NAME and LAST_NAME elements to the local (XQuery engine internal) variable such as $p_name. Upper case is used to normalize names.

5. Leave the return clause as $customer so that all elements in the type are returned.

6. Confirm that the code is as follows (namespaces may be different for your application):

Figure 19-8 Source View of XQueries.ds

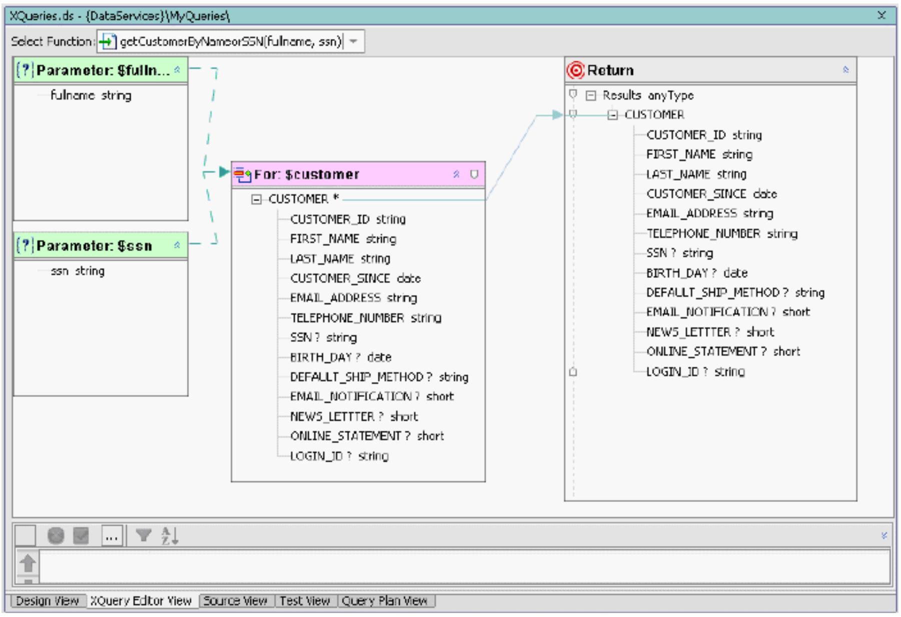


```
XQueries.ds* - {DataServices}\MyQueries\
declare namespace ns1="ld:DataServices/CustomerDB/CUSTOMER";
import schema namespace ns0="ld:DataServices/MyQueries/XQueries" at "ld:DataServices/MyQueries/XQueries.xsd";
declare namespace tns="ld:DataServices/MyQueries/XQueries";

(::pragma  function <f:function kind="read" xmlns:f="urn:annotations.ld.bea.com">
    <uiProperties>
        <component h="150" w="160" y="0" x="0" minimized="false" identifier="fullname"/>
        <component h="150" w="160" y="160" x="0" minimized="false" identifier="ssn"/>
        <component h="406" w="267" y="0" x="501" minimized="false" identifier="returnNode"/>
        <component h="299" w="232" y="95" x="198" minimized="false" identifier="customer"/>
    </uiProperties>
</f:function>
::)
declare function tns:getCustomerByNameorSSN($fullname as xs:string, $ssn as xs:string) as element(ns0:Results
    <tns:Results>
    {
    for $customer in (ns1:CUSTOMER())
     where fn:contains(fn:upper-case(fn:concat($customer/FIRST_NAME,"
            ",$customer/LAST_NAME)), fn:upper-case($fullname) ) or
            fn:starts-with($customer/SSN, $ssn)
     return ($customer)
    }
    </tns:Results>
};
```

Design View | XQuery Editor View | Source View | Test View | Query Plan View

7. Open `XQueries.ds` in XQuery Editor View.

Figure 19-9 Query Editor View of XQueries.ds



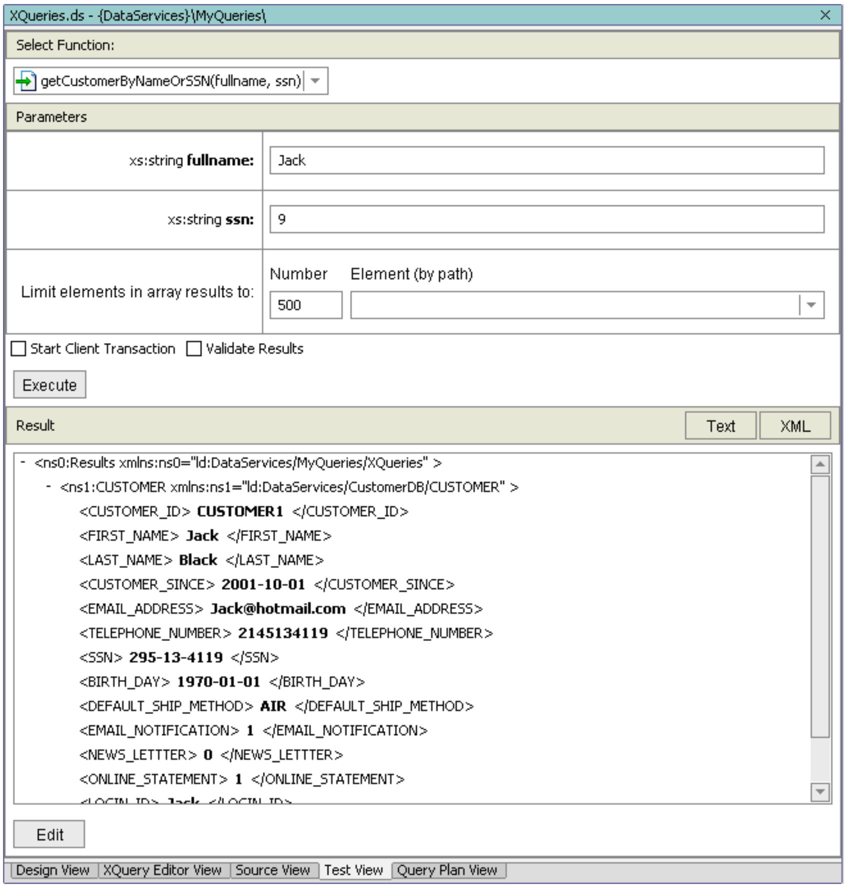
8. Test the query by completing the following steps:

    a. Open `XQueries.ds` in Test View.

    b. Enter a value in both Parameter fields. Neither field can be blank; however, because of the query logic, only one parameter needs to be matched.

    c. Click Execute. The query should return results based on your keyword search parameters. See below for results in Test View and the underlying code.

Figure 19-10 Test Results of String Function



xs:string **fullname:**	Jack
xs:string **ssn:**	9
Limit elements in array results to:	Number 500 Element (by path)

```
- <ns0:Results xmlns:ns0="ld:DataServices/MyQueries/XQueries" >
   - <ns1:CUSTOMER xmlns:ns1="ld:DataServices/CustomerDB/CUSTOMER" >
        <CUSTOMER_ID> CUSTOMER1 </CUSTOMER_ID>
        <FIRST_NAME> Jack </FIRST_NAME>
        <LAST_NAME> Black </LAST_NAME>
        <CUSTOMER_SINCE> 2001-10-01 </CUSTOMER_SINCE>
        <EMAIL_ADDRESS> Jack@hotmail.com </EMAIL_ADDRESS>
        <TELEPHONE_NUMBER> 2145134119 </TELEPHONE_NUMBER>
        <SSN> 295-13-4119 </SSN>
        <BIRTH_DAY> 1970-01-01 </BIRTH_DAY>
        <DEFAULT_SHIP_METHOD> AIR </DEFAULT_SHIP_METHOD>
        <EMAIL_NOTIFICATION> 1 </EMAIL_NOTIFICATION>
        <NEWS_LETTTER> 0 </NEWS_LETTTER>
        <ONLINE_STATEMENT> 1 </ONLINE_STATEMENT>
        <LOGIN_ID> Jack </LOGIN_ID>
```

XQuery Code Reference for a String Function

```
declare function tns:getCustomerByNameOrSSN($fullname as xs:string, $ssn as
xs:string) as element(ns0:Results) {

    <ns0:Results>

    {

        for $customer in (ns1:CUSTOMER())

        where ( fn:contains(fn:upper-case(fn:concat($customer/FIRST_NAME,"
            ",$customer/LAST_NAME)), fn:upper-case($fullname) ) or
            fn:starts-with($customer/SSN, $ssn) )
        return
            ($customer)
    }
</ns0:Results>
```

19.4 Building an Outer Join and Using Order By

Outer joins allow you to get results from the joined objects even if the primary key is not represented in both objects. For example, an outer join of customers and orders reports all customers—even those without orders.

Objectives

In this exercise, you will:

- Build a query that retrieves all customers and lists their addresses, if any.
- Shape the return data to include:
  - All customers, even those without known addresses.
  - Nest addresses with customers (there may be more than 1).
  - Order customers by first name and last name.
  - Order the addresses by zip code.
- Test the function.

Instructions

Note: Namespaces may differ for your application.

1. Add a new function to XQueries.ds and name it getCustomerAddresses.
2. Open `XQueries.ds` in Source View.
3. Define the function declaration by removing the asterisk (*). The code should be as:

   ```
   declare function tns:getCustomerAddresses() as element(ns0:Results) {
   ```

4. Click the + symbol next to the `getCustomerAddresses()` function. This opens the function body.

5. Split the `<tns0:RESULTS/>` element into open and end tags, with curly braces in between for the XQuery.

6. Open XQuery Construct Palette, and then drag and drop the FOR construct between the curly braces. The code should be as follows:

```
for $var in ()

order by ()

return ()
```

7. Set the for clause, using a $customer variable that is associated with `CUSTOMER()` located in the `CustomerDB\CUSTOMER.ds` folder within the Data Services Palette.

```
for $customer in (ns1:CUSTOMER())
```

8. Set the order by clause, by replacing the (), as follows:

```
$customer/FIRST_NAME, $customer/LAST_NAME
```

9. Set the return clause, by replacing the (), as follows:

```
return

<CUSTOMER>

<FIRST_NAME>{fn:data($customer/FIRST_NAME) }</FIRST_NAME>

<LAST_NAME>{fn:data($customer/LAST_NAME)}</LAST_NAME>

  {

    for $address in ()

    where ($address/CUSTOMER_ID eq $customer/CUSTOMER_ID)

    order by $address/ZIPCODE ascending

    return

    $address

  }

  </CUSTOMER>
```

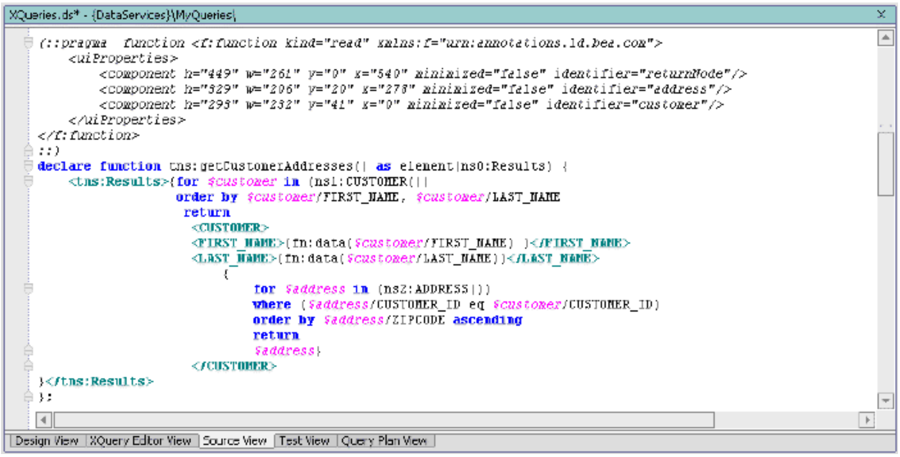
Note: You can either type the code in, or use the XQuery Function Palette and XQuery Construct Palette to build up your query function.

10. Set the $address clause by associating it with `ADDRESS()`, which is located in `CustomerDB\ADDRESS.ds` folder within Data Services Palette.

```
for $address in (ns2:ADDRESS())
```

11. Confirm that the query is as shown in Figure 19-11; namespaces may be different for your application.

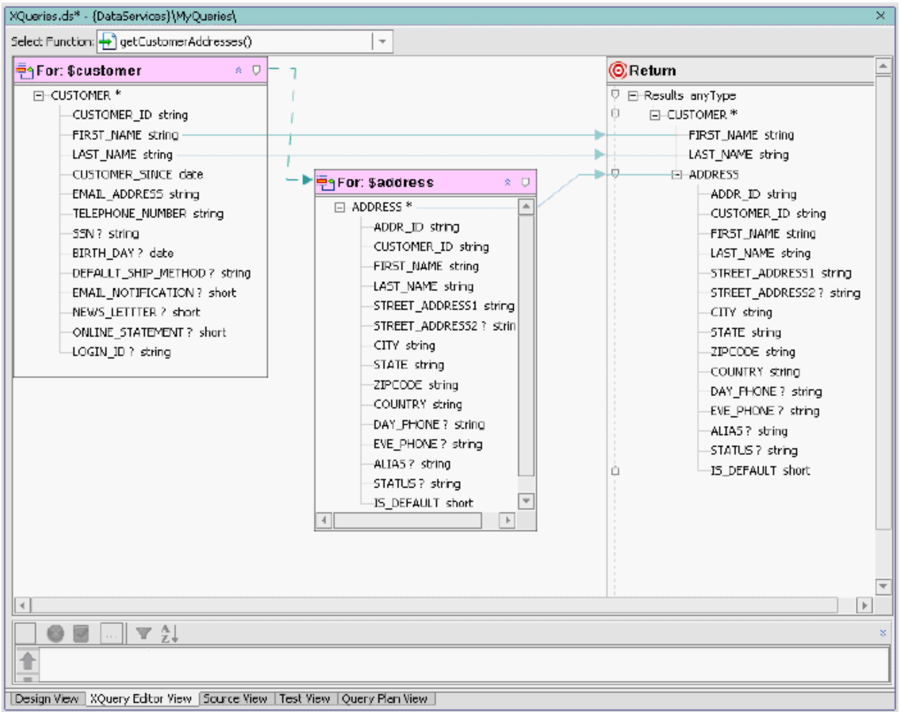
Figure 19-11 Source View of Outer View and Order By Function



```
(::pragma  function <f:function kind="read" xmlns:f="urn:annotations.ld.bea.com">
    <uiProperties>
        <component h="449" w="261" y="0" x="540" minimized="false" identifier="returnNode"/>
        <component h="329" w="206" y="20" x="278" minimized="false" identifier="address"/>
        <component h="293" w="232" y="41" x="0" minimized="false" identifier="customer"/>
    </uiProperties>
</f:function>
::)
declare function tns:getCustomerAddresses() as element(ns0:Results) {
    <tns:Results>{for $customer in (ns1:CUSTOMER()
                  order by $customer/FIRST_NAME, $customer/LAST_NAME
                   return
                    <CUSTOMER>
                    <FIRST_NAME>{fn:data($customer/FIRST_NAME) }</FIRST_NAME>
                    <LAST_NAME>{fn:data($customer/LAST_NAME)}</LAST_NAME>
                        {
                            for $address in (ns2:ADDRESS())
                            where ($address/CUSTOMER_ID eq $customer/CUSTOMER_ID)
                            order by $address/ZIPCODE ascending
                            return
                            $address}
                    </CUSTOMER>
}</tns:Results>
};
```

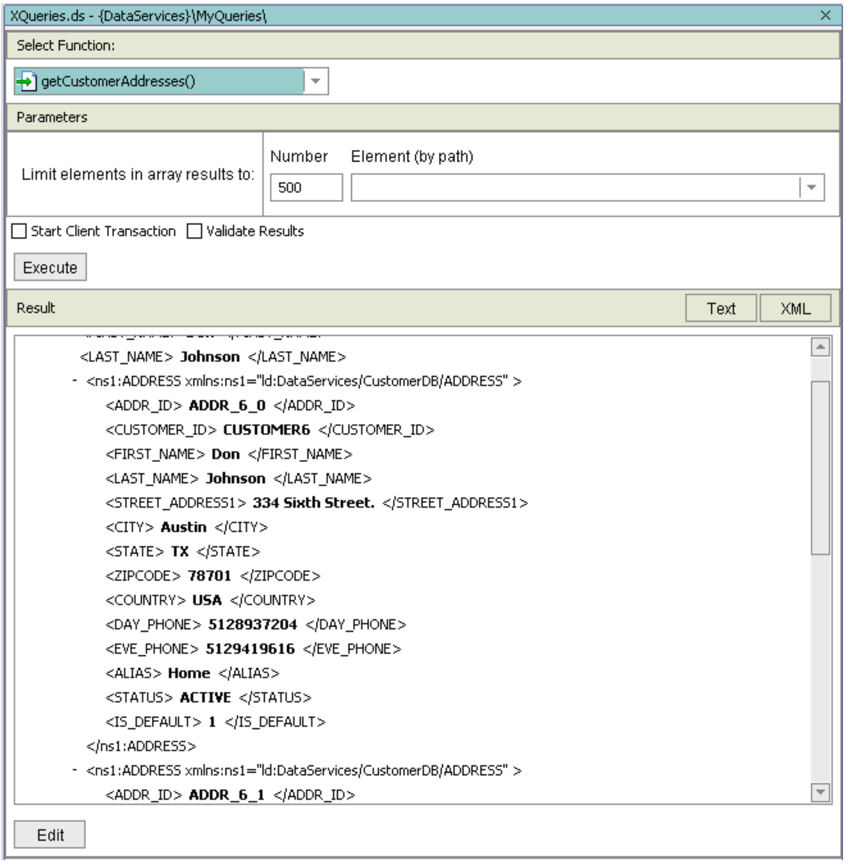
12. Open `XQueries.ds` in XQuery Editor View.

Figure 19-12 XQuery Editor View of Outer Join and Order By Function



13. Open `XQueries.ds` in Test View and test the query; no parameters are required. The XQuery function appears below.

Figure 19-13 Test Results of Outer Join and Order By Function



```
    <LAST_NAME> Johnson </LAST_NAME>
  - <ns1:ADDRESS xmlns:ns1="ld:DataServices/CustomerDB/ADDRESS" >
      <ADDR_ID> ADDR_6_0 </ADDR_ID>
      <CUSTOMER_ID> CUSTOMER6 </CUSTOMER_ID>
      <FIRST_NAME> Don </FIRST_NAME>
      <LAST_NAME> Johnson </LAST_NAME>
      <STREET_ADDRESS1> 334 Sixth Street. </STREET_ADDRESS1>
      <CITY> Austin </CITY>
      <STATE> TX </STATE>
      <ZIPCODE> 78701 </ZIPCODE>
      <COUNTRY> USA </COUNTRY>
      <DAY_PHONE> 5128937204 </DAY_PHONE>
      <EVE_PHONE> 5129419616 </EVE_PHONE>
      <ALIAS> Home </ALIAS>
      <STATUS> ACTIVE </STATUS>
      <IS_DEFAULT> 1 </IS_DEFAULT>
    </ns1:ADDRESS>
  - <ns1:ADDRESS xmlns:ns1="ld:DataServices/CustomerDB/ADDRESS" >
      <ADDR_ID> ADDR_6_1 </ADDR_ID>
```

XQuery Code Reference for an Outer Join and Order By Function

```
declare function tns:getCustomerAddresses() as element(ns0:Results) {

    <tns0:Results>

    {

    for $customer in (ns1:CUSTOMER())

    order by $customer/FIRST_NAME, $customer/LAST_NAME
    return
       <CUSTOMER>
            <FIRST_NAME>{ fn:data($customer/FIRST_NAME) }</FIRST_NAME>
            <LAST_NAME>{fn:data($customer/LAST_NAME)}</LAST_NAME>
            {
                for $address in (ns2:ADDRESS())
                where ($address/CUSTOMER_ID eq $customer/CUSTOMER_ID)
                order by $address/ZIPCODE ascending
                return
                $address
            }
        </CUSTOMER>
    }
    </tns0:Results>
```

19.5 Creating an Inner Join and a Top N

Inner joins mandate that the only items that are returned are with a corresponding entry (such as a primary key in the relational world) in another data source. The following are introduced:

- let clauses
- Nested for clauses
- `concat()` and `subsequence()` XQuery functions

Objectives

In this exercise, you will:

- Build a query that retrieves the top 10 customers who have placed orders with the company.
- Define the shape of the returned data to include:
  - Customer's full name.

  – Order ID.
  – Total order amount (in descending order).

- Test the function.

Instructions

Note: Namespaces may differ for your application.

1. Add a new function to `XQueries.ds` and name it getTop10Customers.
2. Open `XQueries.ds` in Source View.
3. Define the function declaration by removing the asterisk (*). The code should be as follows:

   ```
   declare function tns:getTop10Customers() as element(ns0:Results) {
   ```

4. Click the + symbol next to the `getTop10Customers()` function. This opens the function body.
5. Add curly braces between the two tags.
6. After the opening curly brace, add the following let clause, which will hold the results of subsequent for clauses:

   ```
   let $top10:=
   ```

7. Open XQuery Construct Palette and then drag and drop the FLWOR construct after the let clause. The code should be as follows:

   ```
   for $var in ()

   where true()

   order by ()

   return ()
   ```

8. Set the for clause using a $customer variable that is associated with `CUSTOMER()` located in the `CustomerDB\CUSTOMER.ds` folder within Data Services Palette.

   ```
   for $customer in (ns1:CUSTOMER())
   ```

9. Create a second for clause, using a $order variable that is associated with `CUSTOMER_ORDER()` located in the `ElectronicsDB\CUSTOMER_ORDER.ds` folder within Data Services Palette.

   ```
   for $order in (ns3:CUSTOMER_ORDER())
   ```

10. Set the where clause, by replacing the `true()` with the following code:

```
where ($customer/CUSTOMER_ID eq $order/CUSTOMER_ID)
```

11. Set the order by clause, by entering the following code in the `()`:

```
order by $order/TOTAL_ORDER_AMOUNT descending
```

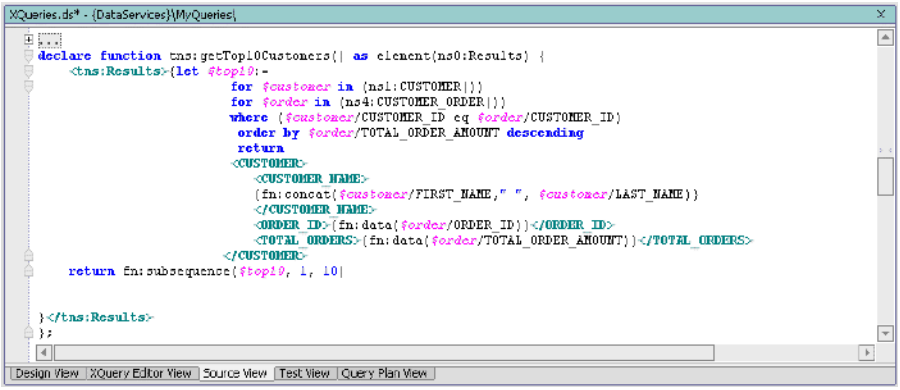
12. Set the return clause, by entering the following code:

```
return
<CUSTOMER>
<CUSTOMER_NAME>
{fn:concat($customer/FIRST_NAME," ", $customer/LAST_NAME)}
     </CUSTOMER_NAME>
     <ORDER_ID>{fn:data($order/ORDER_ID)}</ORDER_ID>
<TOTAL_ORDERS>{fn:data($order/TOTAL_ORDER_AMOUNT)}</TOTAL_ORDERS>
  </CUSTOMER>
return fn:subsequence($top10, 1, 10)
```

Note: You can either type the code in, or use the XQuery Function Palette and XQuery Construct Palette to build up your query.

13. Confirm that the source code is similar to that displayed in Figure 19-14; namespaces may vary.

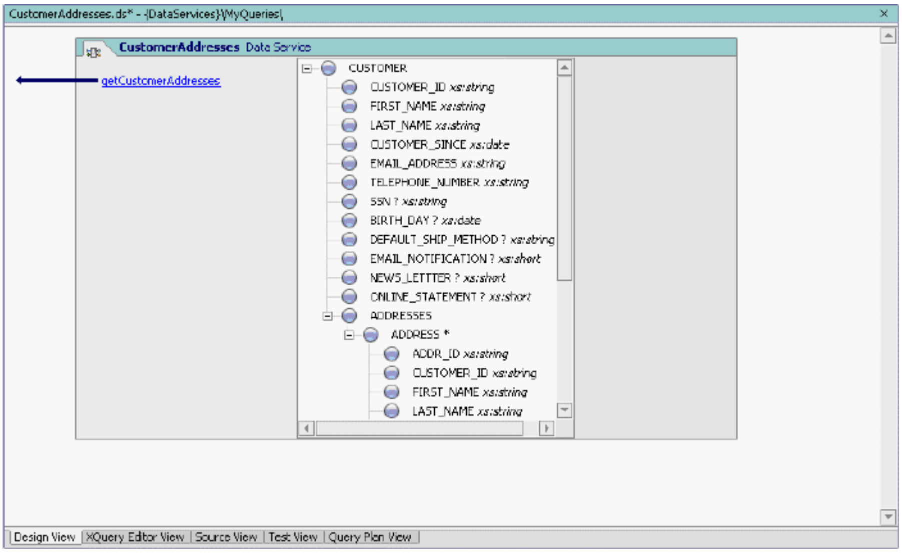
Figure 19-14 Source Code for Inner Join and Top N Function



```
...
declare function tns:getTop10Customers() as element(ns0:Results) {
    <tns:Results>{let $top10:=
                            for $customer in (ns1:CUSTOMER())
                            for $order in (ns4:CUSTOMER_ORDER())
                            where ($customer/CUSTOMER_ID eq $order/CUSTOMER_ID)
                             order by $order/TOTAL_ORDER_AMOUNT descending
                             return
                            <CUSTOMER>
                               <CUSTOMER_NAME>
                               {fn:concat($customer/FIRST_NAME," ", $customer/LAST_NAME)}
                               </CUSTOMER_NAME>
                               <ORDER_ID>{fn:data($order/ORDER_ID)}</ORDER_ID>
                               <TOTAL_ORDERS>{fn:data($order/TOTAL_ORDER_AMOUNT)}</TOTAL_ORDERS>
                           </CUSTOMER>
    return fn:subsequence($top10, 1, 10)


}</tns:Results>
};
```

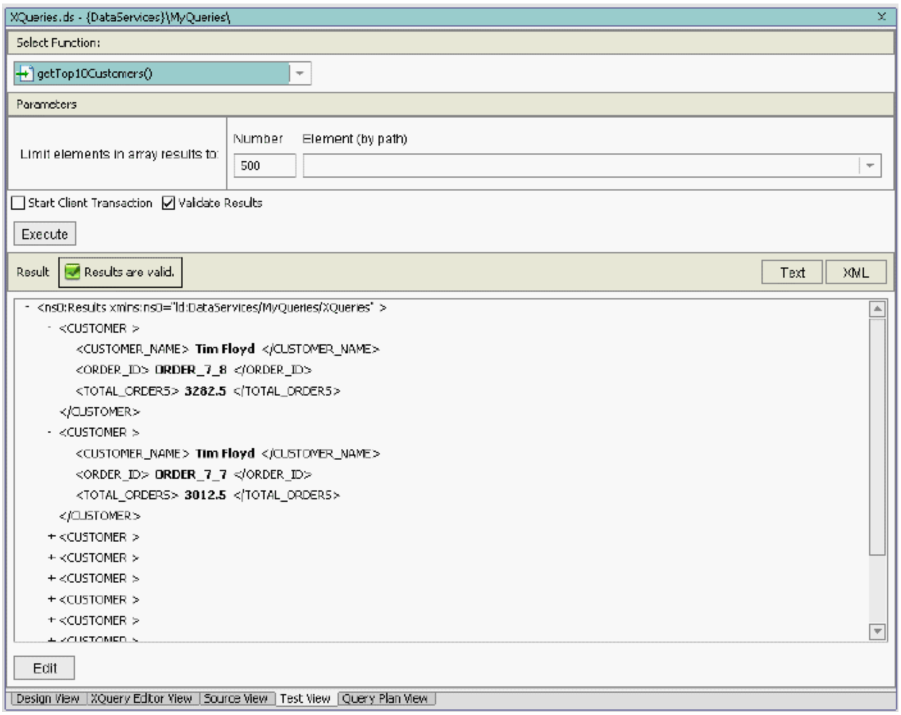
14. Open `XQueries.ds` in XQuery Editor View.

Figure 19-15 XQuery Editor View of Inner Join and Top N Function



15. Open `XQueries.ds` in Test View; no parameters are required to run your query. You should see a document containing the top 10 orders will appear, ordered by total amount. The XQuery function appears below.

Figure 19-16 Test View for Inner Join and Top N Function



```
- <ns0:Results xmlns:ns0="ld:DataServices/MyQueries/XQueries" >
  - <CUSTOMER >
      <CUSTOMER_NAME> Tim Floyd </CUSTOMER_NAME>
      <ORDER_ID> ORDER_7_8 </ORDER_ID>
      <TOTAL_ORDERS> 3282.5 </TOTAL_ORDERS>
    </CUSTOMER>
  - <CUSTOMER >
      <CUSTOMER_NAME> Tim Floyd </CUSTOMER_NAME>
      <ORDER_ID> ORDER_7_7 </ORDER_ID>
      <TOTAL_ORDERS> 3012.5 </TOTAL_ORDERS>
    </CUSTOMER>
  + <CUSTOMER >
  + <CUSTOMER >
  + <CUSTOMER >
  + <CUSTOMER >
  + <CUSTOMER >
```

XQuery Code Reference for Inner Join and Top N Function

```
declare function tns:getTop10Customers() as element(ns0:Results) {

    <tns0:Results>

    {

    let $top10:=

        for $customer in (ns1:CUSTOMER())

        for $order in (ns3:CUSTOMER_ORDER())

        where ($customer/CUSTOMER_ID eq $order/CUSTOMER_ID)

         order by $order/TOTAL_ORDER_AMOUNT descending

         return
```

```
            <CUSTOMER>

                    <CUSTOMER_NAME>

                     {fn:concat($customer/FIRST_NAME," ", $customer/LAST_NAME)}

                    </CUSTOMER_NAME>

                    <ORDER_ID>{fn:data($order/ORDER_ID)}</ORDER_ID>

                         <TOTAL_ORDERS>
                         {fn:data($order/TOTAL_ORDER_AMOUNT)}
                         </TOTAL_ORDERS>

                </CUSTOMER>

             return fn:subsequence($top10, 1, 10)

        }

        </tns0:Results>
```

19.6 Creating a Multi-Level Group By

Retrieving customers grouped by states and cities is not only often needed; it is also a classic database exercise. The following are introduced:

- Group by clause.
- `count()` function.

Objectives

In this exercise, you will:

- Create a query that determines the number of customers, by state and by city.
- Test the function.

Instructions

1. Add a function to `XQueries.ds` and name it `getNumCustomersByState()`.
2. Open `XQueries.ds` in Source View.
3. Define the function declaration, by removing the asterisk *.

4. Click the + symbol next to the `getNumCustomersByState()` function.
5. Split the `<tns0:Results/>` element into open and end tags, with curly braces in between.
6. Open XQuery Construct Palette and then drag and drop the for-group-return (FGR) construct between the curly braces:

   ```
   for $var in ()
   group $var as $varGroup by () as $var2
   return ()
   ```

7. Set the for and group clauses as follows:

   ```
   for $address in ns2:ADDRESS()
   group $address as $stateGroup by $address/STATE as $state
   ```

   Note: Your source is invalid until you complete the next step.

8. Associate the for clause with `ADDRESS()` located in `CustomerDB\Address.ds` within the Data Services Palette as follows:

   ```
   for $address in ns2:ADDRESS()
   ```

9. Set the return clause, as follows:

```
return
       <state>
              <name>{$state}</name>
                     <number>{fn:count($stateGroup/CUSTOMER_ID)}</number>
{
```

   Note: The clause includes the `fn:count()` built-in function, available from the XQuery Function Palette.

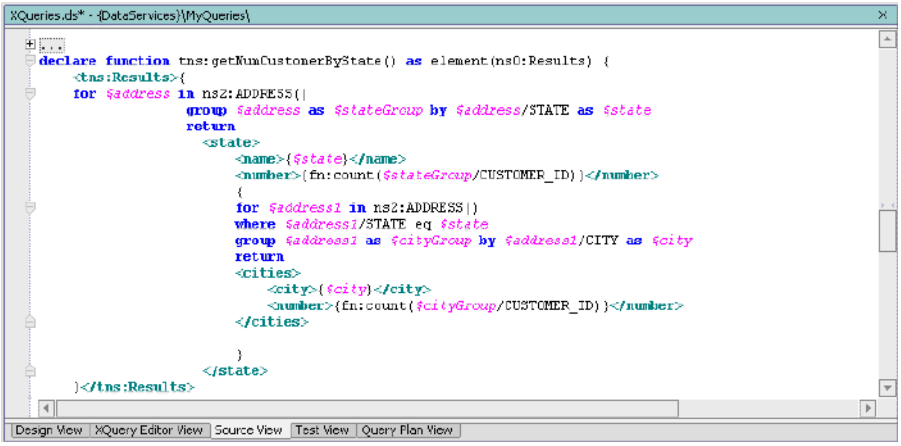
10. Open XQuery Construct Palette and then drag and drop the FWGR construct after the open curly brace of the return clause:

    ```
    for $address1 in ns2:ADDRESS()
    where $address1/STATE eq $state
    group $address1 as $cityGroup by $address1/CITY as $city
    return
    <cities>
    ```

```
<city>{$city}</city>
<number>{fn:count($cityGroup/CUSTOMER_ID)}</number>
</cities>
}
</state>
```

11. Make sure that the namespace in the second for clause is the same as the namespace in the first for clause.
12. Confirm that the code is as displayed in Figure 19-17(namespaces may be different for your application).

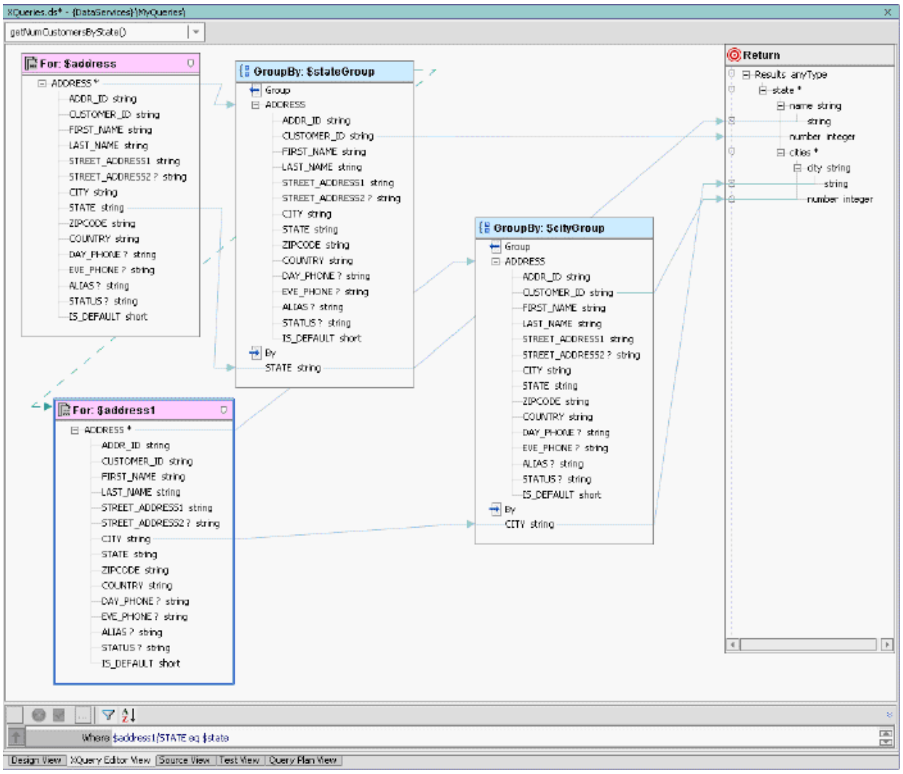
Figure 19-17 Source Code for Multi-Level Group By Function



```
...
declare function tns:getNumCustomerByState() as element(ns0:Results) {
    <tns:Results>{
    for $address in ns2:ADDRESS()
                   group $address as $stateGroup by $address/STATE as $state
                   return
                     <state>
                         <name>{$state}</name>
                         <number>{fn:count($stateGroup/CUSTOMER_ID)}</number>
                         {
                         for $address1 in ns2:ADDRESS()
                         where $address1/STATE eq $state
                         group $address1 as $cityGroup by $address1/CITY as $city
                         return
                         <cities>
                             <city>{$city}</city>
                             <number>{fn:count($cityGroup/CUSTOMER_ID)}</number>
                         </cities>

                         }
                     </state>
    }</tns:Results>
```

13. Open XQueries.ds in XQuery Editor View.

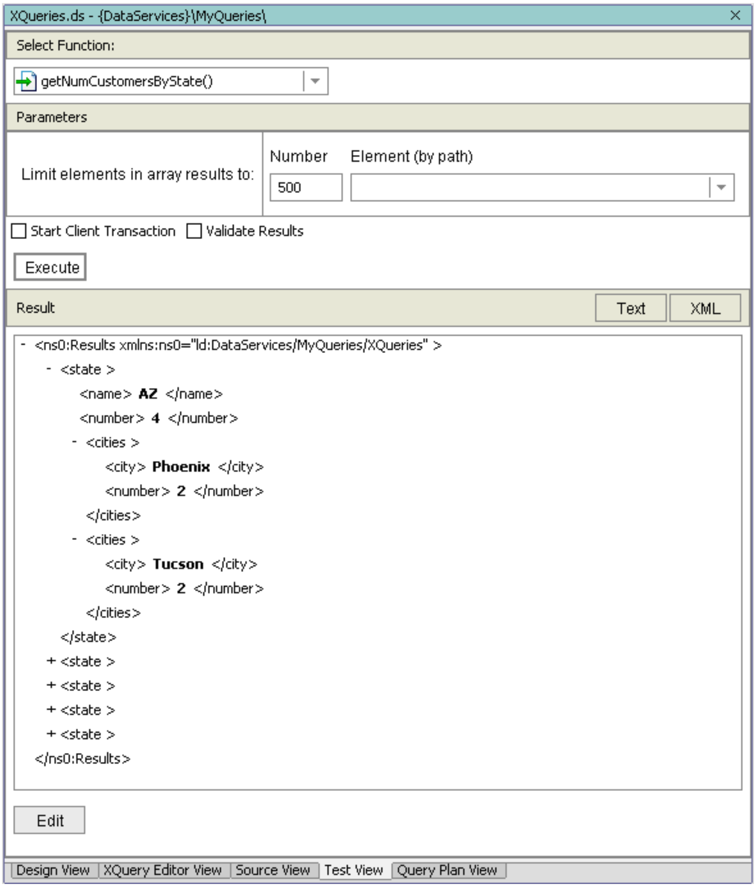
Figure 19-18 XQuery Editor View of Multi-Level Group By Function



Where $address1/STATE eq $state

14. Open XQueries.ds in Test View and test the function; no parameters are required. You should see the state name, followed by the number of customers residing in that state, followed by the city name and number of customers residing in that city. The underlying XQuery also appears below.

Figure 19-19 Test View of Multi-Level Group By Function



```
- <ns0:Results xmlns:ns0="ld:DataServices/MyQueries/XQueries" >
  - <state >
      <name> AZ </name>
      <number> 4 </number>
    - <cities >
        <city> Phoenix </city>
        <number> 2 </number>
      </cities>
    - <cities >
        <city> Tucson </city>
        <number> 2 </number>
      </cities>
    </state>
  + <state >
  + <state >
  + <state >
  + <state >
  </ns0:Results>
```

XQuery Code Reference for Multi-Level Group By Function

```
declare function tns:getNumCustomersByState() as element(ns0:Results) {
  <tns0:Results>
  {
    for $address in ns2:ADDRESS()
    group $address as $stateGroup by $address/STATE as $state
    return
    <state>
        <name>{$state}</name>
        <number>{fn:count($stateGroup/CUSTOMER_ID)}</number>
            {
                for $address1 in ns2:ADDRESS()
                where $address1/STATE eq $state
                group $address1 as $cityGroup by $address1/CITY as $city
                return
                <cities>
                    <city>{$city}</city>
                    <number>{fn:count($cityGroup/CUSTOMER_ID)}</number>
                </cities>
            }
    </state>
}
  </tns0:Results>
  };
```

19.7 Using If-Then-Else If

This example shows how you can create switch-like conditions when building your query. The If-Then-Else-If concept is introduced.

Objectives

In this exercise, you will:

- Create a function that returns different achievement levels as strings for a set of customers, based on their total order amount.
- Test the function.

Instructions

Note: Namespaces may differ for your application.

1. Add a new function to `XQueries.ds` and name it getCustomerLevels.
2. Open `XQueries.ds` in Source View.
3. Define the function declaration, by removing the asterisk (*).
4. Split the `<tns0:Results/>` element into open and end tags, with curly braces ({}) in between.
5. Add a for clause, using a $customer variable that is associated with `CUSTOMER()` located in `CustomerDB\CUSTOMER.ds` within Data Services Palette.

   ```
   for $customer in ns1:CUSTOMER()
   ```

6. Add a second for clause, using an $orders variable that is associated with `CUSTOMER_ORDER()` located in the `ElectronicsDB\CUSTOMER_ORDER.ds` folder within Data Services Palette.

   ```
   for $orders in ns3:CUSTOMER_ORDER()
   ```

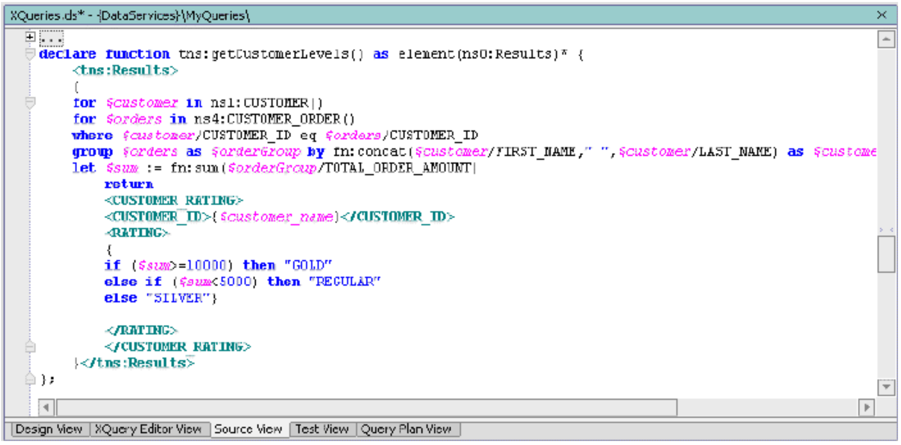
7. Add where, let, and return clause code, placing it immediately after the second for clause:

   ```
   where $customer/CUSTOMER_ID eq $orders/CUSTOMER_ID

   group $orders as $orderGroup by fn:concat($customer/FIRST_NAME,"
   ",$customer/LAST_NAME) as $customer_name

   let $sum := fn:sum($orderGroup/TOTAL_ORDER_AMOUNT)

   return

   <CUSTOMER_RATING>

   <CUSTOMER_ID>{$customer_name}</CUSTOMER_ID>

   <RATING> {if ($sum>=10000) then

    "GOLD"

   else if ($sum<5000) then

   "REGULAR"

   else

   "SILVER"

   }

   </RATING>
   ```

```
</CUSTOMER_RATING>
```

8. Confirm that the code is as displayed in Figure 19-20; namespaces may be different in your application.

Figure 19-20 Source View of If-Then-Else If Function

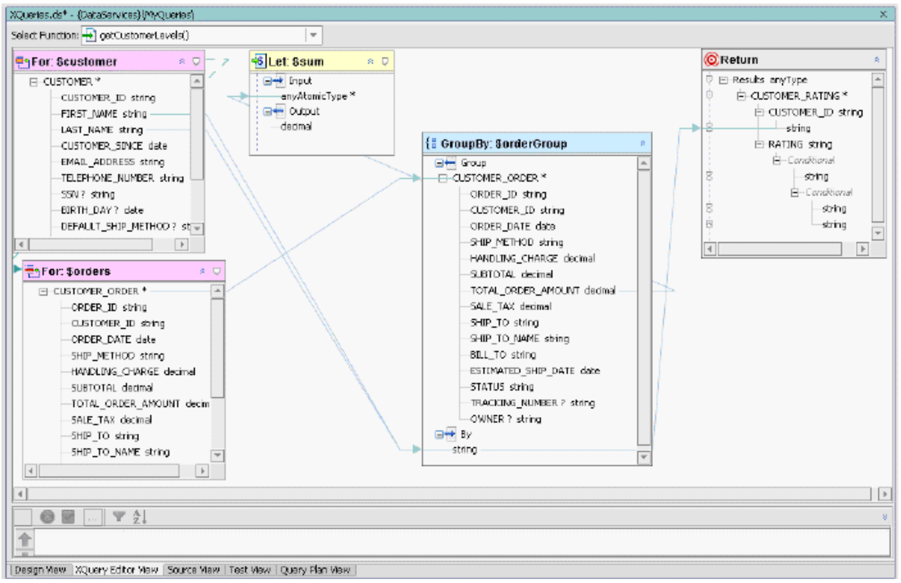


XQueries.ds* - {DataServices}\MyQueries\

```
...
declare function tns:getCustomerLevels() as element(ns0:Results)* {
     <tns:Results>
     {
     for $customer in ns1:CUSTOMER()
     for $orders in ns4:CUSTOMER_ORDER()
     where $customer/CUSTOMER_ID eq $orders/CUSTOMER_ID
     group $orders as $orderGroup by fn:concat($customer/FIRST_NAME," ",$customer/LAST_NAME) as $custome
     let $sum := fn:sum($orderGroup/TOTAL_ORDER_AMOUNT)
          return
          <CUSTOMER_RATING>
          <CUSTOMER_ID>{$customer_name}</CUSTOMER_ID>
          <RATING>
          {
          if ($sum>=10000) then "GOLD"
          else if ($sum<5000) then "REGULAR"
          else "SILVER"}

          </RATING>
          </CUSTOMER_RATING>
     }</tns:Results>
};
```

9. Open `XQueries.ds` in XQuery Editor View.

Figure 19-21 XQuery Editor View of If-Then-Else If Function

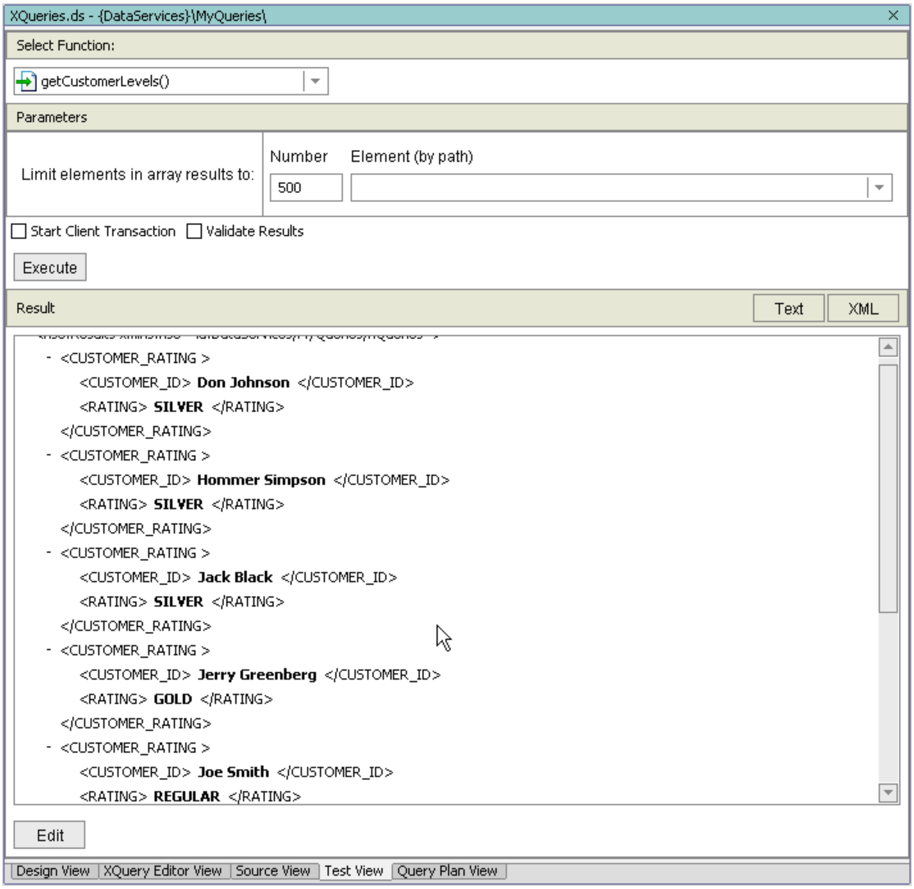


10. Open `XQueries.ds` in Test View and test the function; no parameters are required. When you run the query you will see results organized according to the following levels of purchases:

    - Gold for total orders >= 10000
    - Silver for total orders >= 5000 and <10000
    - Regular for total orders below 5000

    The customer's full name and level are also shown. The XQuery function appears below.

Figure 19-22 Test View of If-Then-Else If Function



getCustomerLevels()

Limit elements in array results to: Number 500 Element (by path)

```
- <CUSTOMER_RATING >
    <CUSTOMER_ID> Don Johnson </CUSTOMER_ID>
    <RATING> SILVER </RATING>
  </CUSTOMER_RATING>
- <CUSTOMER_RATING >
    <CUSTOMER_ID> Hommer Simpson </CUSTOMER_ID>
    <RATING> SILVER </RATING>
  </CUSTOMER_RATING>
- <CUSTOMER_RATING >
    <CUSTOMER_ID> Jack Black </CUSTOMER_ID>
    <RATING> SILVER </RATING>
  </CUSTOMER_RATING>
- <CUSTOMER_RATING >
    <CUSTOMER_ID> Jerry Greenberg </CUSTOMER_ID>
    <RATING> GOLD </RATING>
  </CUSTOMER_RATING>
- <CUSTOMER_RATING >
    <CUSTOMER_ID> Joe Smith </CUSTOMER_ID>
    <RATING> REGULAR </RATING>
```

XQuery Code Reference for If-Then-Else Function

```
declare function tns:getCustomerLevels() as element(ns0:Results) {

  <tns0:Results>

  {

       for $customer in ns1:CUSTOMER()

       for $orders in ns3:CUSTOMER_ORDER()

       where $customer/CUSTOMER_ID eq $orders/CUSTOMER_ID
```

```
        group $orders as $orderGroup by fn:concat($customer/FIRST_NAME," 
    ",$customer/LAST_NAME) as $customer_name

        let $sum := fn:sum($orderGroup/TOTAL_ORDER_AMOUNT)

        return

               <CUSTOMER_RATING>

                              <CUSTOMER_ID>{$customer_name}</CUSTOMER_ID>

                              <RATING> {

                                     if ($sum>=10000) then

                                     "GOLD"

                                            else if ($sum<5000) then

                                     "REGULAR"

                                            else

                                     "SILVER"

                              }

                              </RATING>

                      </CUSTOMER_RATING>

    }

    </tns0:Results>

    };
```

19.8 Creating a Union and Concatenation

This example demonstrates how to integrate data from two different data sources and present the results in a single report that lets you view the data source information as two separate variables.

Objectives

In this exercise, you will:

- Create a function that gathers results from two order entry systems: RTLAPPLOMS and RTLELECOMS.

- Test the function.

Instructions

1. Add a new function to `XQueries.ds` and name it getCombinedOrders.
2. Open `XQueries.ds` in Source View.
3. Define the function declaration, by removing the asterisk * and adding the following parameter:

   ```
   $customer_id as xs:string
   ```
4. Split the <ns0:Results/> element into open and end tags, with curly braces ({}) in between.
5. Open XQuery Construct Palette and then drag and drop the FLWR construct between the curly braces.
6. Set the for clause using a $customer variable that is associated with `CUSTOMER()` located in the `CustomerDB\CUSTOMER.ds` folder within Data Services Palette.

   ```
   for $customer in ns1:CUSTOMER()
   ```
7. Set the let clause, using a $applOrder variable that is associated with `CUSTOMER_ORDER()`, which is located in `ApparelDB\CUSTOMER_ORDER.ds` within Data Services Palette.

   ```
   let $applOrder:= for $order1 in ns4:CUSTOMER_ORDER()
   ```
8. Set the where clause as follows:

   ```
   where $customer/CUSTOMER_ID = $order1/C_ID
   ```
9. Set the return clause, as follows:

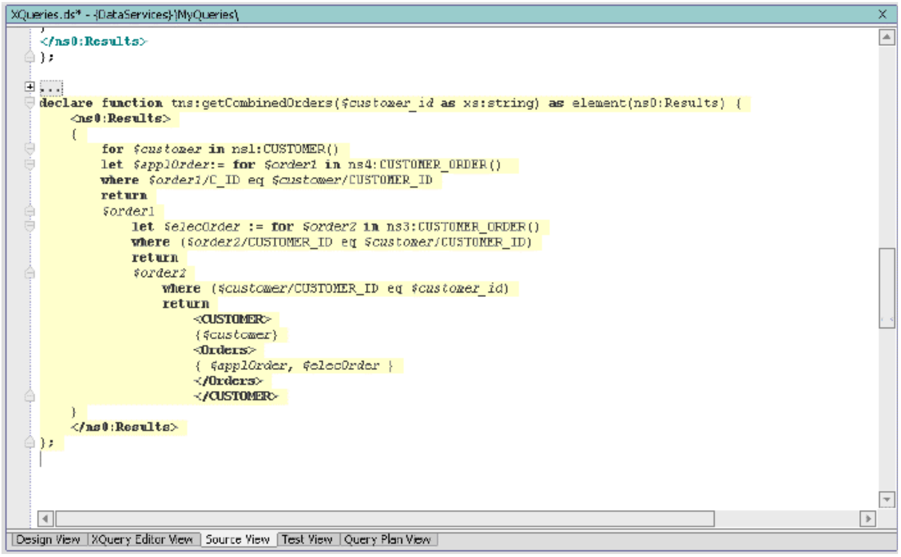
   ```
   return
   $order1
             let $elecOrder := for $order2 in ns3:CUSTOMER_ORDER()
             where ($order2/CUSTOMER_ID eq $customer/CUSTOMER_ID)
             return
             $order2
   where ($customer/CUSTOMER_ID eq $customer_id)
   return
          <CUSTOMER>
          {$customer}
   ```

```
        <Orders>
          {$applOrder, $elecOrder }
        </Orders>
   </CUSTOMER>
```

Note: `ns3:CUSTOMER_ORDER()` refers to `CUSTOMER_ORDER.ds` in ElectronicsDB folder

10. Confirm that the code is as displayed in Figure 19-23; the namespaces may vary in your application.

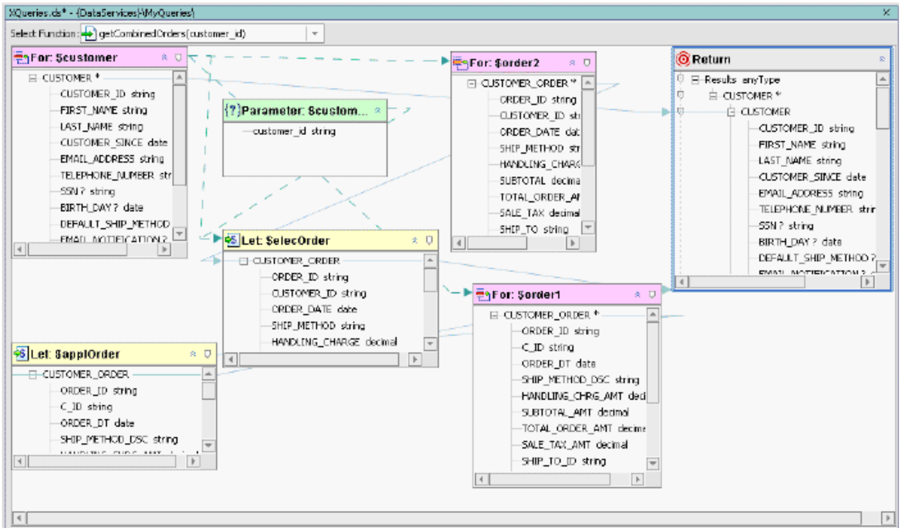
Figure 19-23 Source View for Union and Concatenation Operation



```
</ns0:Results>
};

...
declare function tns:getCombinedOrders($customer_id as xs:string) as element(ns0:Results) {
    <ns0:Results>
    {
        for $customer in ns1:CUSTOMER()
        let $applOrder:= for $order1 in ns4:CUSTOMER_ORDER()
        where $order1/C_ID eq $customer/CUSTOMER_ID
        return
        $order1
            let $elecOrder := for $order2 in ns3:CUSTOMER_ORDER()
            where ($order2/CUSTOMER_ID eq $customer/CUSTOMER_ID)
            return
            $order2
                where ($customer/CUSTOMER_ID eq $customer_id)
                return
                    <CUSTOMER>
                    {$customer}
                    <Orders>
                    { $applOrder, $elecOrder }
                    </Orders>
                    </CUSTOMER>
    }
    </ns0:Results>
};
```

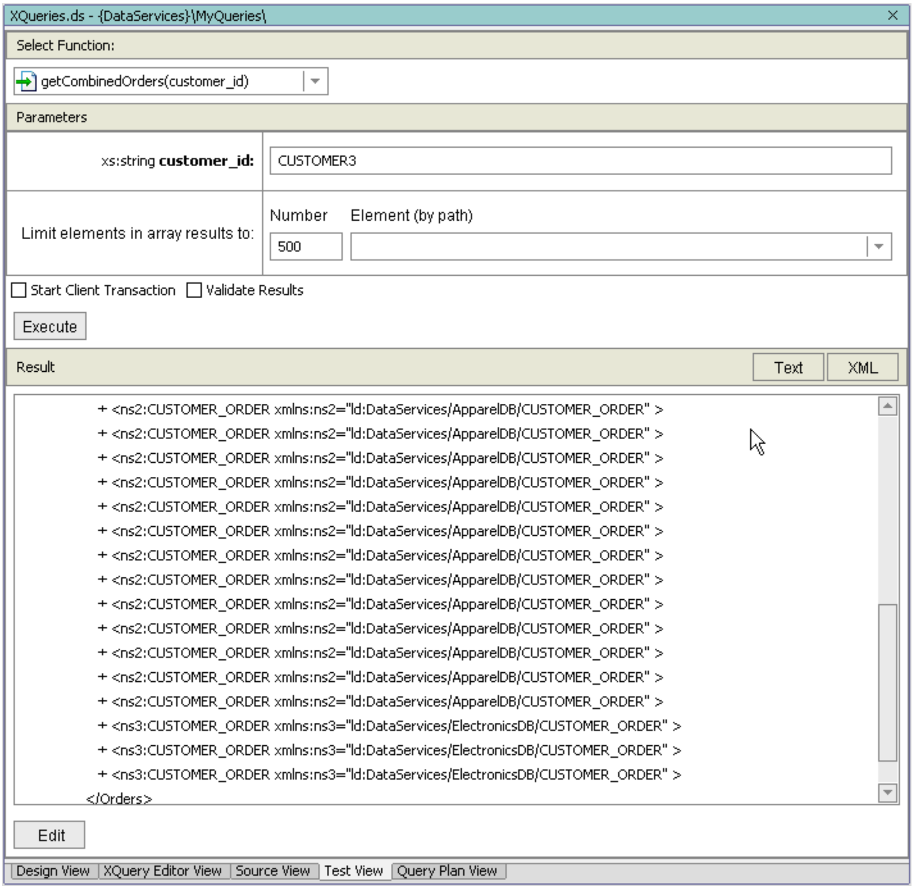
11. Open `XQueries.ds` in XQuery Editor View.

Figure 19-24 XQuery Editor View of Union and Concatenation Operation



12. Open `XQueries.ds` in Test View, and then test the `getCombinedOrders()` function using CUSTOMER3 as the parameter. The XQuery function appears below.

Figure 19-25 Test View of Union and Concatenation Function


XQueries.ds - {DataServices}\MyQueries\
Select Function:
getCombinedOrders(customer_id)
Parameters
xs:string customer_id: CUSTOMER3
Limit elements in array results to: Number 500 Element (by path)
Start Client Transaction
Validate Results
Execute
Result
Text
XML
Edit
Design View
XQuery Editor View
Source View
Test View
Query Plan View

```
        + <ns2:CUSTOMER_ORDER xmlns:ns2="ld:DataServices/ApparelDB/CUSTOMER_ORDER" >
        + <ns2:CUSTOMER_ORDER xmlns:ns2="ld:DataServices/ApparelDB/CUSTOMER_ORDER" >
        + <ns2:CUSTOMER_ORDER xmlns:ns2="ld:DataServices/ApparelDB/CUSTOMER_ORDER" >
        + <ns2:CUSTOMER_ORDER xmlns:ns2="ld:DataServices/ApparelDB/CUSTOMER_ORDER" >
        + <ns2:CUSTOMER_ORDER xmlns:ns2="ld:DataServices/ApparelDB/CUSTOMER_ORDER" >
        + <ns2:CUSTOMER_ORDER xmlns:ns2="ld:DataServices/ApparelDB/CUSTOMER_ORDER" >
        + <ns2:CUSTOMER_ORDER xmlns:ns2="ld:DataServices/ApparelDB/CUSTOMER_ORDER" >
        + <ns2:CUSTOMER_ORDER xmlns:ns2="ld:DataServices/ApparelDB/CUSTOMER_ORDER" >
        + <ns2:CUSTOMER_ORDER xmlns:ns2="ld:DataServices/ApparelDB/CUSTOMER_ORDER" >
        + <ns2:CUSTOMER_ORDER xmlns:ns2="ld:DataServices/ApparelDB/CUSTOMER_ORDER" >
        + <ns2:CUSTOMER_ORDER xmlns:ns2="ld:DataServices/ApparelDB/CUSTOMER_ORDER" >
        + <ns2:CUSTOMER_ORDER xmlns:ns2="ld:DataServices/ApparelDB/CUSTOMER_ORDER" >
        + <ns2:CUSTOMER_ORDER xmlns:ns2="ld:DataServices/ApparelDB/CUSTOMER_ORDER" >
        + <ns3:CUSTOMER_ORDER xmlns:ns3="ld:DataServices/ElectronicsDB/CUSTOMER_ORDER" >
        + <ns3:CUSTOMER_ORDER xmlns:ns3="ld:DataServices/ElectronicsDB/CUSTOMER_ORDER" >
        + <ns3:CUSTOMER_ORDER xmlns:ns3="ld:DataServices/ElectronicsDB/CUSTOMER_ORDER" >
      </Orders>
```

XQuery Reference Code for Union and Concatenation Operation

```
declare function tns:getCombinedOrders($customer_id as xs:string) as
element(ns0:Results) {

    <tns0:Results>

    {

        for $customer in ns1:CUSTOMER()
        let $applOrder:= for $order1 in ns4:CUSTOMER_ORDER()
           where ($order1/C_ID eq $customer/CUSTOMER_ID)
           return
           $order1
        let $elecOrder := for $order2 in ns3:CUSTOMER_ORDER()
           where ($order2/CUSTOMER_ID eq $customer/CUSTOMER_ID)
           return
           $order2
        where ($customer/CUSTOMER_ID eq $customer_id)
        return
           <CUSTOMER>
              {$customer}
              <Orders>
                { $applOrder, $elecOrder }
              </Orders>
           </CUSTOMER>
    }
    </tns0:Results>
  };
```

Lesson Summary

In this lesson you, learned how to:

- Use Source View to add, edit, or delete XQuery code that defines a data service's query functions.
- Compare the coded query with the XQuery Editor View.

Implementing Relationship Functions and Logical Modeling

Relationship functions return data combined from two or more data services. For example, by creating a relationship between the Address and Customer data services, you can obtain the address for a given customer. Or by creating a relationship between the Customer and Order Management data services, you can receive data that identifies all orders returned by a particular customer.

Model diagrams are used to view a selected set of data services and the relationships between them. The model shows the basic structure of the data returned by the data service. The main purpose of the diagram is to help you envision meaningful subsets of your enterprise data relationships, but it can also be used to define new artifacts or edit existing artifacts.

Logical modeling is an extension of the physical modeling that you learned about in Tutorial 5: Modeling Data Services. There are three exercises in this lesson, which are to be completed in sequential order. The exercises in this tutorial are dependent on the work completed in the previous tutorials.

Objectives

After completing this lesson, you will be able to:

- Create model diagrams for a logical data service.
- Define relationships between data services.
- View and implement multiple relationship functions.
- Test multiple relationship functions.

Overview

To help you get from a complex, distributed physical data landscape to a more holistic view of enterprise information, ALDSP supports a visual, model-driven approach to developing data services. Modeling provides a graphical representation of the data resources in your environment, providing a bird's-eye view of a large system or giving you a way to create "zoomed" views of enterprise areas. In a model diagram data services appear as boxes, while relationships appear as annotated lines connection the data service representations. A relationship is only visible if both end points are also on the diagram.

The result is real-time access to externally persisted data through a logical data model.

20.1 Implementing and Testing a Relationship Function

The `getCustomer_Order()` function is intended to return customer order information for a specific customer. However, to accomplish that you need to add the ApparelDB data service's CUSTOMER_ORDER as a source schema, and then create a relationship with the target schema.

Objectives

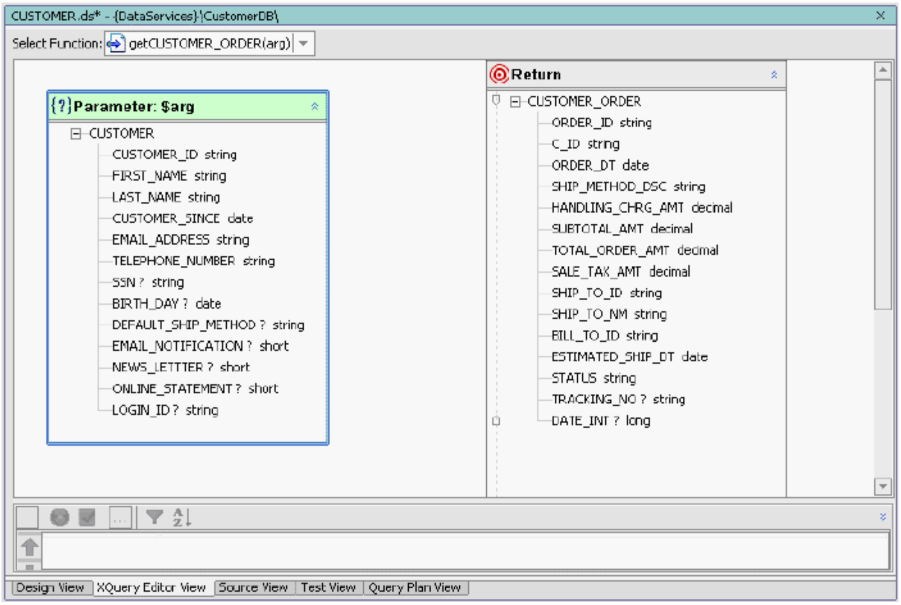
In this exercise you will:

- Implement a relationship function, using XQuery Editor View to define the return data service, by:
  - Identifying the data source.
  - Creating an overwrite map between source and target elements.
  - Creating a simple map between a parameter and a source element.
- Test the relationship function created as a result of the mappings.

Instructions

1. Open `CUSTOMER.ds` in XQuery Editor View. The file is located in DataServices\CustomerDB.
2. Select `getCustomer_Order(arg)` from the Function drop-down list.

Figure 20-1 XQuery Editor View of getCustomer_Order Function



3. In Data Services Palette, expand the ApparelDB and CUSTOMER_ORDER.ds folders.

4. Drag and drop `CUSTOMER_ORDER()` into XQuery Editor View.

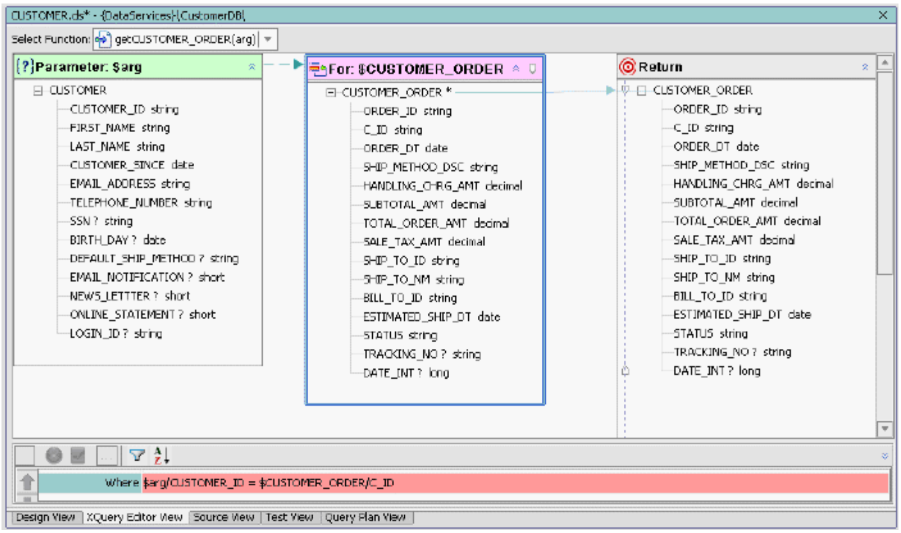
5. In XQuery Editor View, create an overwrite mapping between the CUSTOMER_ORDER source and Return elements by completing the following steps:

   a. Press Ctrl.

   b. Drag and drop the source node's CUSTOMER_ORDER* element onto the Return type's CUSTOMER_ORDER element.

6. Drag and drop the parameter's CUSTOMER_ID element onto the source node's C_ID element. Confirm that the `getCustomer_Order()` function is as displayed in Figure 20-2.

Figure 20-2 Joined and Mapped Function


CUSTOMER.ds* - {DataServices}-\CustomerDB\

Select Function: getCUSTOMER_ORDER(arg)

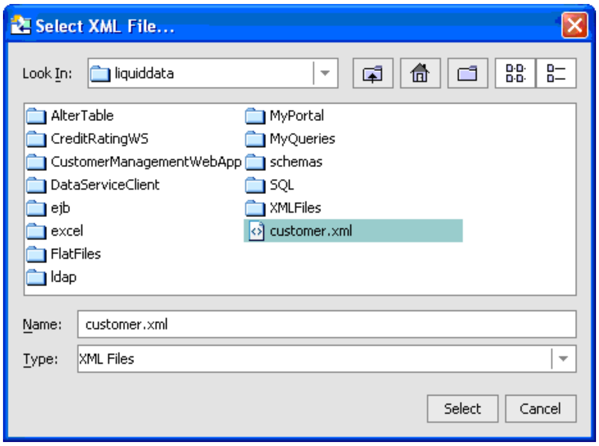
```
Parameter: $arg
  CUSTOMER
    CUSTOMER_ID string
    FIRST_NAME string
    LAST_NAME string
    CUSTOMER_SINCE date
    EMAIL_ADDRESS string
    TELEPHONE_NUMBER string
    SSN ? string
    BIRTH_DAY ? date
    DEFAULT_SHIP_METHOD ? string
    EMAIL_NOTIFICATION ? short
    NEWS_LETTTER ? short
    ONLINE_STATEMENT ? short
    LOGIN_ID ? string

For: $CUSTOMER_ORDER
  CUSTOMER_ORDER *
    ORDER_ID string
    C_ID string
    ORDER_DT date
    SHIP_METHOD_DSC string
    HANDLING_CHRG_AMT decimal
    SUBTOTAL_AMT decimal
    TOTAL_ORDER_AMT decimal
    SALE_TAX_AMT decimal
    SHIP_TO_ID string
    SHIP_TO_NM string
    BILL_TO_ID string
    ESTIMATED_SHIP_DT date
    STATUS string
    TRACKING_NO ? string
    DATE_INT ? long

Return
  CUSTOMER_ORDER
    ORDER_ID string
    C_ID string
    ORDER_DT date
    SHIP_METHOD_DSC string
    HANDLING_CHRG_AMT decimal
    SUBTOTAL_AMT decimal
    TOTAL_ORDER_AMT decimal
    SALE_TAX_AMT decimal
    SHIP_TO_ID string
    SHIP_TO_NM string
    BILL_TO_ID string
    ESTIMATED_SHIP_DT date
    STATUS string
    TRACKING_NO ? string
    DATE_INT ? long
```

Where $arg/CUSTOMER_ID = $CUSTOMER_ORDER/C_ID

7. Save your work and then build the DataServices project.
8. Open `CUSTOMER.ds` in Test View and run a test by completing the following steps:

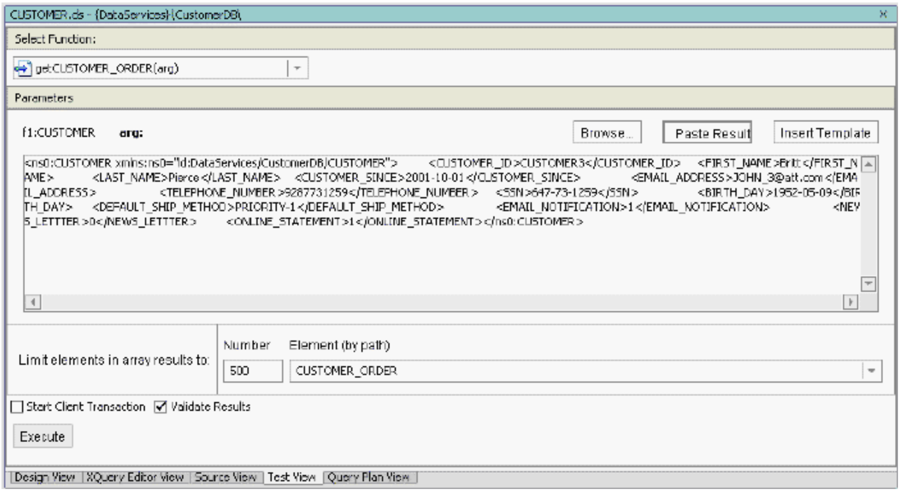
- Select `getCUSTOMER_ORDER(arg)` from the Function drop-down list.
- Click Browse, navigate to, and open the `<beahome>\weblogic81\samples\LiquidData\EvalGuide` directory.
- Select the customer.xml file.

Figure 20-3 Select XML File



9. Click Select. The contents of the file are inserted into the Parameters field.

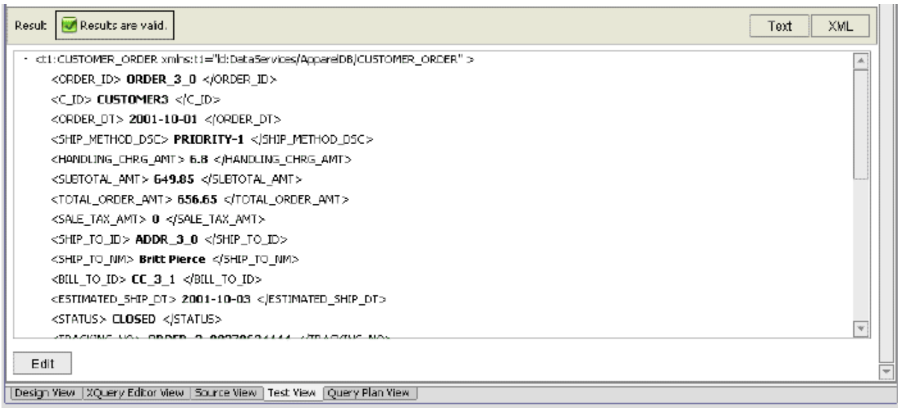
Figure 20-4 Select XML File



```
<ns0:CUSTOMER xmlns:ns0="ld:DataServices/CustomerDB/CUSTOMER">    <CUSTOMER_ID>CUSTOMER3</CUSTOMER_ID>    <FIRST_NAME>Britt</FIRST_N
AME>    <LAST_NAME>Pierce</LAST_NAME>    <CUSTOMER_SINCE>2001-10-01</CUSTOMER_SINCE>    <EMAIL_ADDRESS>JOHN_3@att.com</EMA
IL_ADDRESS>    <TELEPHONE_NUMBER>9287731259</TELEPHONE_NUMBER>    <SSN>647-73-1259</SSN>    <BIRTH_DAY>1952-05-09</BIR
TH_DAY>    <DEFAULT_SHIP_METHOD>PRIORITY-1</DEFAULT_SHIP_METHOD>    <EMAIL_NOTIFICATION>1</EMAIL_NOTIFICATION>    <NEW
S_LETTTER>0</NEWS_LETTTER>    <ONLINE_STATEMENT>1</ONLINE_STATEMENT></ns0:CUSTOMER>
```

10. Click Execute. The order information for CUSTOMER3 should appear.

Figure 20-5 Relationship Test Results



```
- <t1:CUSTOMER_ORDER xmlns:t1="ld:DataServices/ApparelDB/CUSTOMER_ORDER" >
    <ORDER_ID> ORDER_3_0 </ORDER_ID>
    <C_ID> CUSTOMER3 </C_ID>
    <ORDER_DT> 2001-10-01 </ORDER_DT>
    <SHIP_METHOD_DSC> PRIORITY-1 </SHIP_METHOD_DSC>
    <HANDLING_CHRG_AMT> 6.8 </HANDLING_CHRG_AMT>
    <SUBTOTAL_AMT> 649.85 </SUBTOTAL_AMT>
    <TOTAL_ORDER_AMT> 656.65 </TOTAL_ORDER_AMT>
    <SALE_TAX_AMT> 0 </SALE_TAX_AMT>
    <SHIP_TO_ID> ADDR_3_0 </SHIP_TO_ID>
    <SHIP_TO_NM> Britt Pierce </SHIP_TO_NM>
    <BILL_TO_ID> CC_3_1 </BILL_TO_ID>
    <ESTIMATED_SHIP_DT> 2001-10-03 </ESTIMATED_SHIP_DT>
    <STATUS> CLOSED </STATUS>
```

20.2 Creating a Model Diagram for Logical Data Services

Model diagrams display the basic structure of the data returned by a data service. A model diagram lets you view a selected set of data services and the relationships between them. The main purpose of the diagram is to help you envision meaningful subsets of the model, but it can also be used to define new artifacts or edit existing artifacts.

Objectives

In this exercise, you will:

- Import a schema that provides a logical and unified representation of two separate physical data sources.
- Create a basic model diagram by adding data services to the imported logical data service.
- Create relationship functions between the modeled data services.

Instructions

1. Import the OrderManagement schema into the DataServices project folder by completing the following steps:
   a. Right-click the DataServices project folder.
   b. Choose Import.

   c. Navigate to and open the `<beahome>\weblogic81\samples\LiquidData\EvalGuide directory`.

   d. Select the OrderManagement folder.

   e. Click Import. A new folder, OrderManagement, is created in the DataServices project. The imported schema contains logical representations of the two Order Management Systems (Apparel and Electronics), which make the two systems appear as if they are a single Order Management System.

2. Create a sub-folder within the Models folder by completing the following steps:

   a. Right-click the MODELS folder, located in the DataServices folder.

   b. Choose New → Folder.

   c. Enter Logical in the Name field.

   d. Click OK.

3. Create a new logical model diagram by completing the following steps:

   a. Right-click the Logical folder.

   b. Choose New → Model Diagram.

   c. Enter `OrderManagement_Logical_Model.md` in the Name field.

   d. Click Create.

4. Create a model for the OrderManagement data services by completing the following steps:

   a. Expand the CustomerManagement, OrderManagement, and ServiceDB folders.

   b. Drag and drop the following `.ds` files into the model:

Table 20-6 Model data services

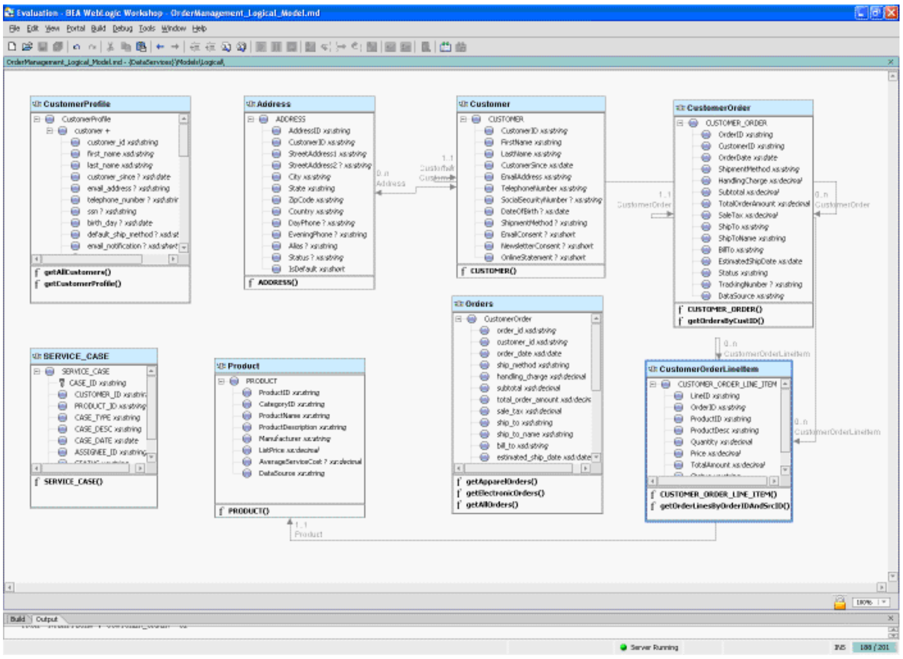
Data Service File	Located in:
`customerProfile.ds`	CustomerManagement
`address.ds`	OrderManagement
`Customer.ds`	OrderManagement
`customerOrder.ds`	OrderManagement

Table 20-6 Model data services

Data Service File	Located in:
`customerOrderLineItem.ds`	OrderManagement
`orders.ds`	OrderManagement
`product.ds`	OrderManagement
`service_case.ds`	ServiceDB

Your model diagram should be similar to that displayed in Figure 20-7. Notice that relationships between data services already exist. These relationships were generated during the Import Source Metadata process, and are based on the foreign key relationship defined in the underlying relational data.

Figure 20-7 Model Diagram for Logical Data Services



5. Create a relationship between the CustomerProfile and ADDRESS data services by completing the following steps:

   a. Drag and drop the `customer_id element (CustomerProfile)` onto the CustomerID element (Address).

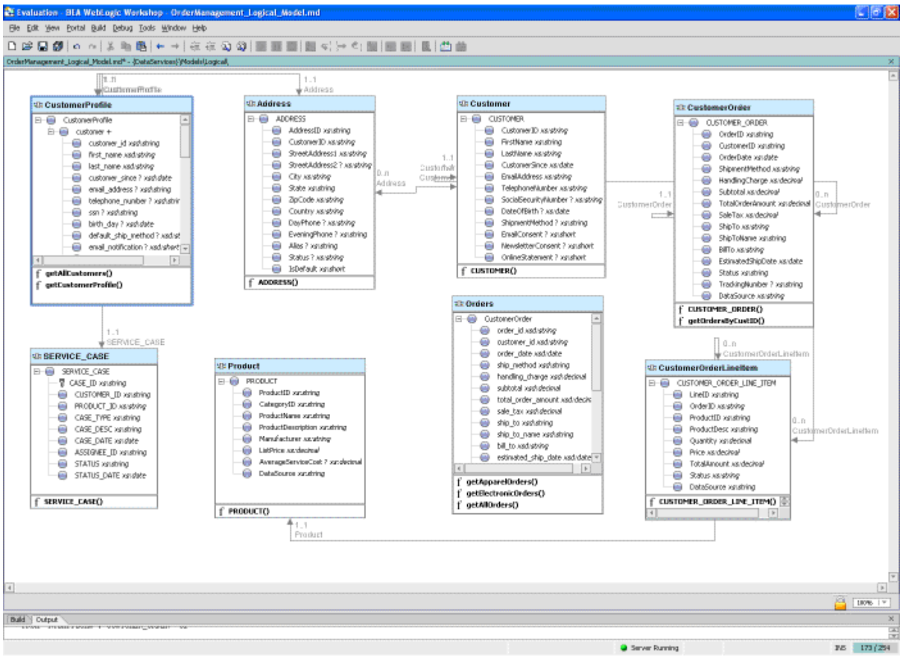
   b. Click Finish in the Relationship Properties window.

6. Create a relationship between CustomerProfile and SERVICE_CASE data services by completing the following steps:

   a. Drag and drop the `customer_id (CustomerProfile)` onto the CUSTOMER_ID element (SERVICE_CASE).

   b. Click Finish in the Relationship Properties window.

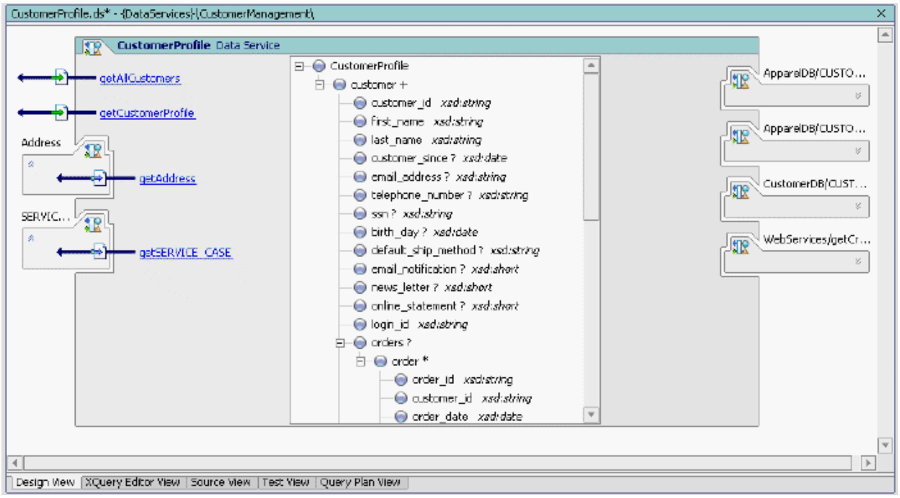
Figure 20-8 New Relationships Defined



7. Open CustomerProfile.ds in Design View. You should see two new relationship functions, `getAddress()` (which navigates to the Address logical data service, located in

OrderManagement) and `getSERVICE_CASE()` (which navigates to the SERVICE_CASE physical data service, located in ServiceDB).

Figure 20-9 New Functions



8. Save your work.

Lesson Summary

In this exercise, you learned to:

- Import a schema that provides a logical and unified representation of two separate physical data sources.
- Create a basic model diagram by adding data services to the imported logical data service.
- Create relationship functions between the modeled data services.

Running Ad Hoc Queries

Sometimes it is necessary to execute a query on functions associated with an application that is already deployed. Rather than take the application offline to create a new query, ALDSP provides the PreparedExpression class, which lets you create and run ad hoc queries on deployed applications.

Objectives

After completing this lesson, you will be able to:

- Create an ad hoc query from within a ALDSP application.
- Run an ad hoc query.

Overview

ALDSP includes a PreparedExpression class that lets you build an ad hoc query using remote data sources, and then execute it using the Mediator API or ALDSP Control. Using the methods within the PreparedExpression class, you can build queries on top of existing XDS functions belonging to applications already deployed on an active local or remote server domain.

The process for running an ad hoc query is as follows:

1. Create a StringBuffer to hold the query.
2. Create an instance of the PreparedExpression class, using the prepareExpression method.
3. Create parameters for the ad hoc query, using the bind<DataType> methods.

4. Submit the query and review the results, using the Mediator API or ALDSP Control.

21.1 Creating an Instance of the PreparedExpression Class

The first steps in creating an ad hoc query are to instantiate a StringBuffer and the PreparedExpression class. For the latter instance, you use the prepareExpression method of the DataServiceFactory class, which accepts three parameters:

- InitialContext
- Application Name
- XQuery String

For example:

```
PreparedExpression pe = DataServiceFactory.prepareExpression(
getInitialContext(),
"Evaluation",
xquery.toString()
     );
```

Objectives

In this exercise, you will:

- Build a StringBuffer instance to hold the ad hoc query.
- Create an instance of the PreparedExpression class.

Instructions

1. Create a new Java project in the Evaluation application, and name it AdHocClient.
2. Create a new Java class in the AdHocClient project, and name it AdHocQuery.
3. Open AdHocQuery.java.
4. Import the following Java classes:

   ```
   import com.bea.ld.dsmediator.client.DataServiceFactory;
   ```

```
import com.bea.ld.dsmediator.client.PreparedExpression;
import com.bea.xml.XmlObject;
import javax.naming.InitialContext;
import javax.naming.NamingException;
import javax.xml.namespace.QName;
import weblogic.jndi.Environment;
```

Note: You can also import the necessary Java classes by first adding the code specified below, and then pressing Alt + Enter.

5. Specify the initial context for the query, by adding the following code after the first curly brace:

```
public static InitialContext getInitialContext() throws NamingException
{
   Environment env = new Environment();
    env.setProviderUrl("t3://localhost:7001");

env.setInitialContextFactory("weblogic.jndi.WLInitialContextFactory");
    env.setSecurityPrincipal("weblogic");
    env.setSecurityCredentials("weblogic");
    return new InitialContext(env.getInitialContext().getEnvironment());
  }
```

6. Add the main argument, by adding the following code after the initial context:

```
public static void main (String args[]) {
System.out.println("========== Ad Hoc Client =============");
try {
} catch (Exception e) {
e.printStackTrace();
}
  }
```

7. Build a StringBuffer instance to hold your query. For example, add the following code after the line:

```
   try {:
StringBuffer xquery = new StringBuffer();
```

```
xquery.append("declare variable $p_firstname as xs:string external;
\n");

xquery.append("declare variable $p_lastname as xs:string external;  \n");


xquery.append("declare namespace
ns1=\"ld:DataServices/MyQueries/XQueries\"; \n");

xquery.append("declare namespace
ns0=\"ld:DataServices/CustomerDB/CUSTOMER\"; \n\n");


xquery.append("<ns1:RESULTS>                                        \n");

xquery.append("{                                                    \n");

xquery.append("    for $customer in ns0:CUSTOMER()                  \n");

xquery.append("    where ($customer/FIRST_NAME eq $p_firstname  \n");

xquery.append("       and $customer/LAST_NAME eq $p_lastname)   \n");

xquery.append("    return                                           \n");

xquery.append("      $customer                                      \n");

xquery.append(" }                                                   \n");

xquery.append("</ns1:RESULTS>                                       \n");
```

8. Use the prepareExpression method of the Mediator API's DataServiceFactory class to create an instance of the PreparedExpression class, by adding the following code:

```
PreparedExpression pe = DataServiceFactory.prepareExpression(

getInitialContext(), "Evaluation",  xquery.toString());
```

21.2 Defining Ad Hoc Query Parameters

After you create an instance of the PreparedExpression class, you need to specify the parameters that will be passed when the ad hoc query is submitted. To pass parameters, you use one or more bind<DataType> methods, such as bindString and bindInt.

Objectives

In this exercise, you will:

- Use the bind<DataType> methods of the PreparedExpression instance to pass parameters.

- Invoke the query.
- Display the query's XML results.

Instructions

1. Pass parameters by using the bindString method of the PreparedExpression instance. For example, add the following code to the `AdHocQuery.java` file:

   ```
   pe.bindString(new QName("p_firstname"), "Jack");
   pe.bindString(new QName("p_lastname"), "Black");
   ```

2. Invoke the executeQuery method to return the query results in an XmlObject.

   ```
   XmlObject obj = pe.executeQuery();
   ```

3. Enter the code necessary to return the XmlObject and display the XML. For example:

   ```
   System.out.println(obj.toString());
   ```

21.3 Testing the Ad Hoc Query

You are now ready to test the ad hoc query, which is set to return information for Jack Black.

Objectives

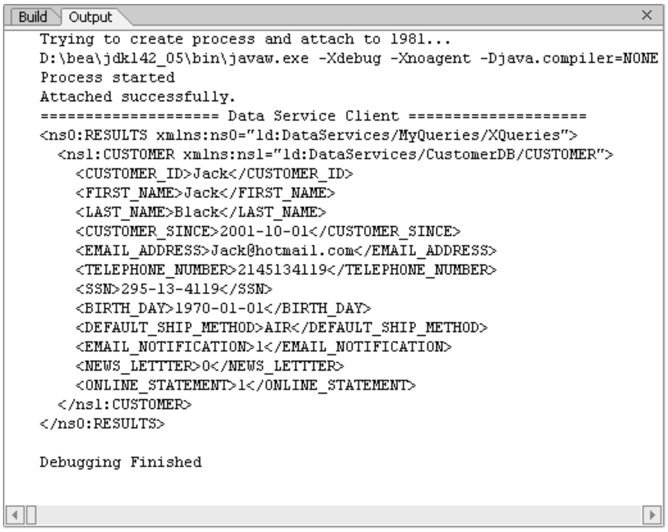
In this exercise, you will:

- Build the AdHocClient project.
- Run the AdHocQuery.java

Instructions

1. Build the AdHocClient project.
2. In the `AdHocQuery.java` application, click the Start icon (or press Ctrl + F5).
3. Confirm that you can retrieve customer profile information for Jack Black.

Figure 21-1 Results of Ad-Hoc Query () Function



```
Trying to create process and attach to 1981...
D:\bea\jdk142_05\bin\javaw.exe -Xdebug -Xnoagent -Djava.compiler=NONE
Process started
Attached successfully.
=================== Data Service Client ====================
<ns0:RESULTS xmlns:ns0="ld:DataServices/MyQueries/XQueries">
  <ns1:CUSTOMER xmlns:ns1="ld:DataServices/CustomerDB/CUSTOMER">
    <CUSTOMER_ID>Jack</CUSTOMER_ID>
    <FIRST_NAME>Jack</FIRST_NAME>
    <LAST_NAME>Black</LAST_NAME>
    <CUSTOMER_SINCE>2001-10-01</CUSTOMER_SINCE>
    <EMAIL_ADDRESS>Jack@hotmail.com</EMAIL_ADDRESS>
    <TELEPHONE_NUMBER>2145134119</TELEPHONE_NUMBER>
    <SSN>295-13-4119</SSN>
    <BIRTH_DAY>1970-01-01</BIRTH_DAY>
    <DEFAULT_SHIP_METHOD>AIR</DEFAULT_SHIP_METHOD>
    <EMAIL_NOTIFICATION>1</EMAIL_NOTIFICATION>
    <NEWS_LETTTER>0</NEWS_LETTTER>
    <ONLINE_STATEMENT>1</ONLINE_STATEMENT>
  </ns1:CUSTOMER>
</ns0:RESULTS>

Debugging Finished
```

Code Reference for an Ad Hoc Query

```
import com.bea.ld.dsmediator.client.DataServiceFactory;

   import com.bea.ld.dsmediator.client.PreparedExpression;

   import com.bea.xml.XmlObject;

   import javax.naming.InitialContext;

   import javax.naming.NamingException;

   import javax.xml.namespace.QName;

   import weblogic.jndi.Environment;


   public class AdHocQuery

   {

  public static InitialContext getInitialContext() throws NamingException {

             Environment env = new Environment();

             env.setProviderUrl("t3://localhost:7001");
env.setInitialContextFactory("weblogic.jndi.WLInitialContextFactory");
              env.setSecurityPrincipal("weblogic");
              env.setSecurityCredentials("weblogic");
              return new
InitialContext(env.getInitialContext().getEnvironment());
   }
     public static void main (String args[]) {
              System.out.println("==================== Ad Hoc Client
====================");
              try {
                     StringBuffer xquery = new StringBuffer();

xquery.append("declare variable $p_firstname as xs:string external; \n");
xquery.append("declare variable $p_lastname as xs:string external;  \n");

xquery.append("declare namespace
ns1=\"ld:DataServices/MyQueries/XQueries\"; \n");
xquery.append("declare namespace
ns0=\"ld:DataServices/CustomerDB/CUSTOMER\"; \n\n");

xquery.append("<ns1:RESULTS>                                    \n");
xquery.append("{                                                \n");
xquery.append("    for $customer in ns0:CUSTOMER()              \n");
xquery.append("    where ($customer/FIRST_NAME eq $p_firstname  \n");
xquery.append("       and $customer/LAST_NAME eq $p_lastname)   \n");
xquery.append("    return                                       \n");
xquery.append("      $customer                                  \n");
xquery.append(" }                                               \n");
xquery.append("</ns1:RESULTS>                                    \n");


PreparedExpression pe =
DataServiceFactory.prepareExpression(getInitialContext(), "Evaluation",
xquery.toString());
pe.bindString(new QName("p_firstname"), "Jack");
pe.bindString(new QName("p_lastname"), "Black");
XmlObject results = pe.executeQuery();
System.out.println(results);


} catch (Exception e) {
                     e.printStackTrace();
              }
  }
}
```

Lesson Summary

In this lesson, you learned how to:

- Create a StringBuffer instance to hold the ad hoc query.
- Create an instance of the PreparedExpression class, using the prepareExpression method of the Mediator API's DataServiceFactory class.
- Create parameters for the ad hoc query, using the bindString method of the PreparedExpression class.
- Submit the query and review the results, using the Mediator API.
- Review the XML output.

Creating Data Services Based on SQL Statements

The SQL-Exit feature lets developers re-use SQL statements that are currently available in the source system. These user-defined SQL statements are bound in XQuery as external functions, in the same manner as all ALDSP sources.

Objectives

After completing this lesson, you will be able to:

- Create data service based on a user-defined SQL statement.
- Use that data service to retrieve customer and address information together.

Overview

Configuring the SQL-exit data source involves the following steps:

1. Create the `.xsd` schema that describes the SQL results.
2. Create the data service, including annotations, describing the result set.
3. Associate an XML Type for the data service to the schema previously created.

When a user-defined SQL statement is used within other functions, the ALDSP engine will bind the SQL statement as a sub-query in a new SQL statement. To disable this functionality, the metadata property is Subquery, stored in the function's pragma, can be set to value false.

22.1 Creating a Data Service from a User-Defined SQL Statement

The SQL statement that will be used to create a new data service involves a join between the CUSTOMER and ADDRESS data services. You need to manually add all the necessary metadata to the new data service, before this query can execute. To do so, you will use metadata previously imported from the CUSTOMER and ADDRESS tables.

Objectives

In this exercise, you will:

- Import an SQL statement as source metadata for a physical data service.
- Generate a new data service.

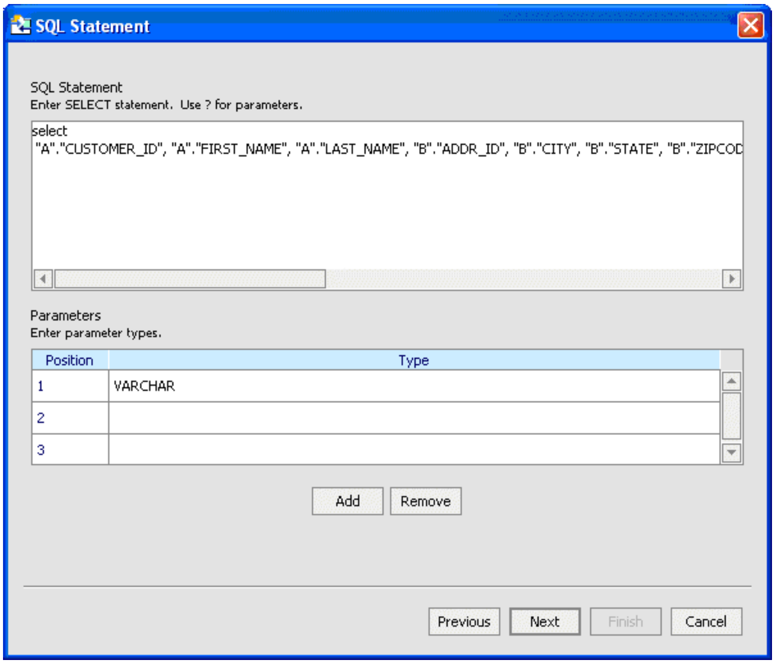
Instructions

1. Open the `SQL_Statement.txt` file, located in the `<beahome>\weblogic81\samples\LiquidData\EvalGuide` folder.
2. Copy the text within the file. The text is:

```
select "A"."CUSTOMER_ID", "A"."FIRST_NAME", "A"."LAST_NAME", "B"."ADDR_ID",
"B"."CITY", "B"."STATE", "B"."ZIPCODE", "B"."COUNTRY" from
"RTLCUSTOMER"."CUSTOMER" "A", "RTLCUSTOMER"."ADDRESS" "B" where
"A"."CUSTOMER_ID" = "B"."CUSTOMER_ID" AND "B"."STATE" = ?
```

3. Create a new folder in the DataServices project and name it SQL. You will use this folder to store a new data service based on user-defined SQL statements.
4. Right-click the SQL folder and select Import Source Metadata.
5. Select Relational from the Data Source Type and click Next.
6. Select the SQL statement radio button and click Next. The SQL Statement page opens.
7. Paste the copied text into the SQL Statement field.
8. Select VARCHAR from the Type column for Position 1 and click Next. The Summary page opens.

Figure 22-1 SQL Statement

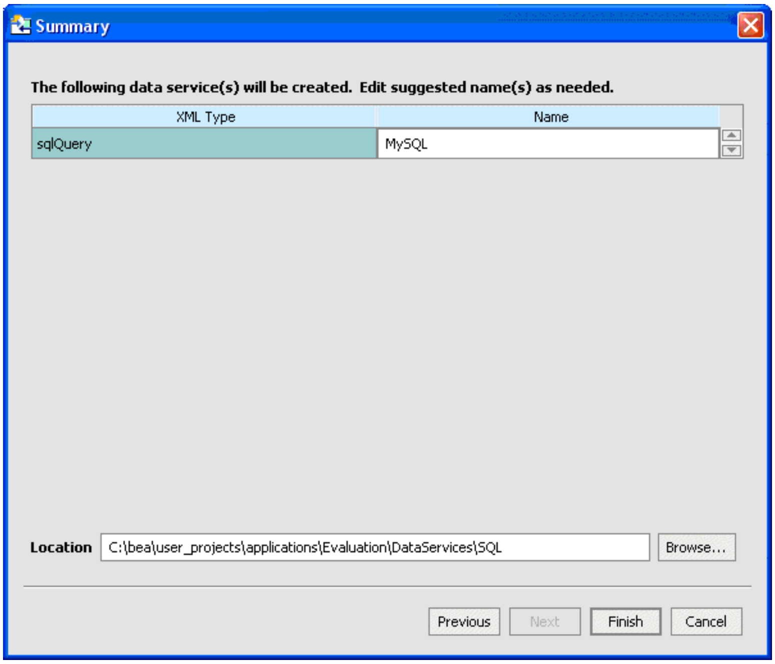


```
select
"A"."CUSTOMER_ID", "A"."FIRST_NAME", "A"."LAST_NAME", "B"."ADDR_ID", "B"."CITY", "B"."STATE", "B"."ZIPCOD
```

Position	Type
1	VARCHAR
2	
3	

9. Rename the data service to MySQL.

Figure 22-2 Summary for SQL-Based Data Service



The following data service(s) will be created. Edit suggested name(s) as needed.

XML Type	Name
sqlQuery	MySQL

Location C:\bea\user_projects\applications\Evaluation\DataServices\SQL

10. Click Finish. The MySQL data service and associated schema files are added to the SQL folder.

22.2 Testing Your SQL Data Service

You are now ready to test whether the MySQL data service can retrieve all customers who reside in California.

Objectives

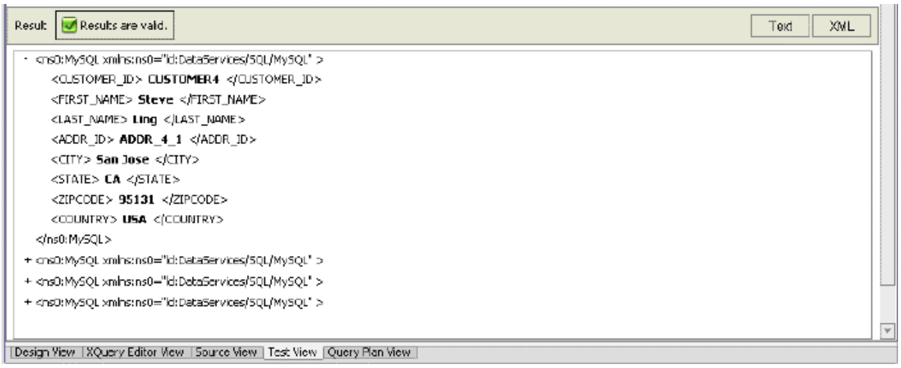
In this exercise, you will:

- Test the MySQL data service.
- View the results.

Instructions

1. Open `MySQL.ds` in Test View.
2. Select `MySQL(x1)` from the Function drop-down list.
3. In the parameter box enter CA
4. Click Execute. The result set will show customer and address information for the state of California.

Figure 22-3 Test Results for an SQL-Based Data Service



```
- <ns0:MySQL xmlns:ns0="ld:DataServices/SQL/MySQL" >
    <CUSTOMER_ID> CUSTOMER4 </CUSTOMER_ID>
    <FIRST_NAME> Steve </FIRST_NAME>
    <LAST_NAME> Ling </LAST_NAME>
    <ADDR_ID> ADDR_4_1 </ADDR_ID>
    <CITY> San Jose </CITY>
    <STATE> CA </STATE>
    <ZIPCODE> 95131 </ZIPCODE>
    <COUNTRY> USA </COUNTRY>
  </ns0:MySQL>
+ <ns0:MySQL xmlns:ns0="ld:DataServices/SQL/MySQL" >
+ <ns0:MySQL xmlns:ns0="ld:DataServices/SQL/MySQL" >
+ <ns0:MySQL xmlns:ns0="ld:DataServices/SQL/MySQL" >
```

Lesson Summary

In this lesson, you learned how to:

- Manually create a data service out of an SQL statement.
- Test the SQL-based data service.

Performing Custom Data Manipulation Using Update Override

ALDSP permits customized updates through the use of the update override feature. The update override logic, which is triggered prior to submitting data, can be used for custom data manipulation, update overrides, logging, debugging, or other custom logic needs.

In this lesson, you will write an update override that computes total orders, based on the quantity and price of each order.

Objectives

After completing this lesson, you will be able to:

- Write customized data manipulation through an update override.
- Associate an update override with a data service.

Overview

An update override, which you assign to a data service, performs custom logic prior to submitting data. The update override is a Java class that implements the com.bea.sdo.mediator.UpdateOverride class. Using that class's performChange (DataGraph graph) method, a Data Graph instance of the current data service is returned. The Data Graph can then be manipulated in using the update override logic.

For example, you can get the CustomerProfileDocument DataObject through the data graph

```
(CustomerProfileDocument) graph.getRootObject();
```

You could also get the Change Logging summary through `graph.getChangeSummary()`

On return of the Data Graph, the following conditions apply:

- Return true: Proceed with the rest of update.
- Return false: Stop the update.
- Throw Exception: Rollback.

23.2 Creating an Update Override

An update override enables custom manipulation of data within data service.

Objectives

In this exercise, you will:

- Create a new Java class that will serve as the basis for an update override.
- Import and implement an update override class.
- Implement the performChange method.
- Write customized update logic.

Instructions

1. Create a new Java class by completing the following steps:
    a. Right-click the CustomerManagement folder, located in the DataServices folder.
    b. Choose New → Java Class.
    c. Enter CustomerProfileExit in the File Name field.
    d. Click Create.
2. Build the DataServices project.
3. Open the `CustomerProfileExit.java` file.
4. Import and implement the update override, by completing the following steps:
    a. Import the update override by entering the following code:

```
import com.bea.ld.dsmediator.update.UpdateOverride;
```

b. Implement the update override by modifying the public class CustomerProfileExit code, as follows:

```
public class CustomerProfileExit implements UpdateOverride
```

c. Press Alt + Enter, and then click OK to add the `performChange(DataGraph)` signature.

d. Implement the `performChange(DataGraph graph)` method by modifying the code to read as follows:

```
public boolean performChange(DataGraph graph)
```

The DataGraph passed in the argument contains the current SDO instance with all changes, including the change summary.

5. Access the update override by casting the root object of the data graph to your SDO. Add the following code, after the opening braces:

```
CustomerProfileDocument customerDocument =
            (CustomerProfileDocument) graph.getRootObject();
```

6. Press Alt+Enter. With this CustomerProfileDocument instance, you can get and set values that will be applied to the SDO before it is submitted.

7. Write update logic to compute the total order amount, based on the sum of each order item's quantity multiplied by its price (sum of price*qty). You can use this to get the total of each item's quantity*price and to set the total order amount to this value.

Note: Use BigDecimals for computations.

For example:

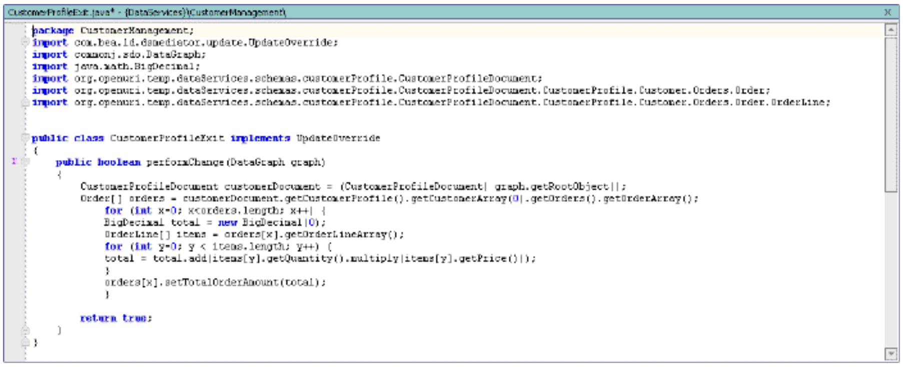
```
Order[] orders =
customerDocument.getCustomerProfile().getCustomerArray(0).getOrders().g
etOrderArray();


for (int x=0; x<orders.length; x++) {

BigDecimal total = new BigDecimal(0);

OrderLine[] items = orders[x].getOrderLineArray();

for (int y=0; y < items.length; y++) {

total =
total.add(items[y].getQuantity().multiply(items[y].getPrice()));

}
orders[x].setTotalOrderAmount(total);
}
```

8. Press Alt + Enter, for all flagged items.
9. Enter the code necessary to return the results. For example:

```
System.out.println(">>> CustomerProfile.ds Exit completed");
return true;
}
}
```

10. Confirm that your code is as displayed in Figure 23-1.
11. Build DataServices project.

Figure 23-1 Update Override Code


CustomerProfileExit.java* - {DataServices}\CustomerManagement\

```
package CustomerManagement;
import com.bea.ld.dsmediator.update.UpdateOverride;
import commonj.sdo.DataGraph;
import java.math.BigDecimal;
import org.openuri.temp.dataServices.schemas.customerProfile.CustomerProfileDocument;
import org.openuri.temp.dataServices.schemas.customerProfile.CustomerProfileDocument.CustomerProfile.Customer.Orders.Order;
import org.openuri.temp.dataServices.schemas.customerProfile.CustomerProfileDocument.CustomerProfile.Customer.Orders.Order.OrderLine;


public class CustomerProfileExit implements UpdateOverride
{
    public boolean performChange(DataGraph graph)
    {
        CustomerProfileDocument customerDocument = (CustomerProfileDocument) graph.getRootObject();
        Order[] orders = customerDocument.getCustomerProfile().getCustomerArray(0).getOrders().getOrderArray();
            for (int x=0; x<orders.length; x++) {
            BigDecimal total = new BigDecimal(0);
            OrderLine[] items = orders[x].getOrderLineArray();
            for (int y=0; y < items.length; y++) {
            total = total.add(items[y].getQuantity().multiply(items[y].getPrice()));
            }
            orders[x].setTotalOrderAmount(total);
            }

        return true;
    }
}
```

23.3 Associating an Update Override to a Logical Data Service

Before you can use the update override, you must associate it with a specific data service.

Objectives

In this exercise you will:

- Use the Property Editor to associate an update override with a specific data service.

- Build the data service to include the update override.

Instructions

1. Open the CustomerProfile data service in Design View.
2. Click the CustomerProfile header to activate the Property Editor. (If the Property Editor is not open, press Alt + 6.)
3. Click the update override class field.
4. Navigate to the `DataServices.jar\CustomerManagement` folder.
5. Select `CustomerProfileExit.class` and click Open. The update override class field is now populated with `CustomerManagement.CustomerProfileExit`.
6. Build the DataServices project.

23.4 Testing the Update Override

As with any other data service, you should test the update override to ensure that it works properly.

Objectives

In this exercise you will:

- Change order information from within your CustomerManagementWebApp application.
- Confirm update override results.

Instructions

1. Open `CustomerPageFlowController.jpf`, which is located in the `CustomerManagementWebApp\CustomerPageFlow` folder.
2. Click the Start icon to open Workshop Test Browser.
3. Enter CUSTOMER3 in the CUSTOMER ID field and click Submit.

   Note: It may take a few seconds before the information is returned.
4. Change the order information by adding, modifying or deleting order lines.
5. Click Submit All Changes.

6. Click Back to return to the CUSTOMER ID page.
7. Enter CUSTOMER3 in the CUSTOMER ID field and click Submit.
8. Confirm if the updated total order information was computed.

```
Update Override Reference Code

   package CustomerManagement;

   import com.bea.ld.dsmediator.update.UpdateOverride;

   import commonj.sdo.DataGraph;

   import java.math.BigDecimal;

   import
org.openuri.temp.dataServices.schemas.customerProfile.CustomerProfileDo
cument;

   import
org.openuri.temp.dataServices.schemas.customerProfile.CustomerProfileDo
cument.CustomerProfile.Customer.Orders.Order;

   import
org.openuri.temp.dataServices.schemas.customerProfile.CustomerProfileDo
cument.CustomerProfile.Customer.Orders.Order.OrderLine;


   public class CustomerProfileExit implements UpdateOverride

   {

    public boolean performChange(DataGraph graph)

    {

        CustomerProfileDocument customerDocument =
(CustomerProfileDocument) graph.getRootObject();

        Order[] orders =
customerDocument.getCustomerProfile().getCustomerArray(0).getOrders().g
etOrderArray();

            for (int x=0; x<orders.length; x++) {

            BigDecimal total = new BigDecimal(0);

            OrderLine[] items = orders[x].getOrderLineArray();

            for (int y=0; y < items.length; y++) {

            total =
total.add(items[y].getQuantity().multiply(items[y].getPrice()));

            }
            orders[x].setTotalOrderAmount(total);
            }
        return true;
    }
      }
```

Lesson Summary

In this lesson, you learned how to:

- Create an update override for a logical data service.
- Write logic in the update override to access the XML bean and perform custom data manipulation prior to submitting.
- Associate an update override to the data service.

Updating Web Services Using Update Override

You can also use update overrides to update a Web service.

Objectives

After completing this lesson, you will be able to:

- Write an update override function for performing manual updates.
- View your results.

Overview

Unlike relational data sources, Web service updates are not automated, because ALDSP is unable to determine how to decompose a read function into a corresponding write. To enable ALDSP to perform the necessary writes, you must create an update override for the physical data service, and then implement the necessary writes in that update override. For example:

```
public class CreditRatingExit implements UpdateOverride {

    public boolean performChange(DataGraph datagraph){


    // don't do anything if there are no changes

        ChangeSummary cs = datagraph.getChangeSummary();

        if (cs.getChangedDataObjects().size()==0)
```

```
            return true;

    // get changed values from SDO
       GetCreditRatingResponseDocument creditRating =
(GetCreditRatingResponseDocument) datagraph.getRootObject();
       int newRating =
creditRating.getGetCreditRatingResponse().getGetCreditRatingResult().getRa
ting();
       String customerId =
creditRating.getGetCreditRatingResponse().getGetCreditRatingResult().getCu
stomerId();

    // update CreditRating web service
       try {
            CreditRatingDBTestSoap ratingWS = new
CreditRatingDBTest_Impl().getCreditRatingDBTestSoap();
            CreditRating rating = new CreditRating(newRating,customerId);
            ratingWS.setCreditRating(rating);
       } catch (Exception e) {
            e.printStackTrace();
            return false;
       }
       System.out.println("WEB SERVICE EXIT COMPLETE!");
       return true;
    }
}
```

24.1 Creating an Update Override for a Physical Data Service

The clientgen utility in WebLogic generates a Web Service-specific client .jar file that client applications can use to invoke Web Services. You simply need to specify the WSDL URI, the name and location of the `client.jar` file to generate and a package structure. Clientgen is available as an ant task as well as a Java application that can be invoked from the command line.

For more information on clientgen see:

`http://e-docs.bea.com/wls/docs81/webserv/anttasks.html`

Objectives

In this exercise, you will:

- Edit the WebLogic clientgen command to point to your WebLogic Server.
- Run the clientgen utility.
- Add the generated client .jar file to your application Library.

Instructions

Set the clientgen command line utility to generate a Web service `client .jar` file by completing the following steps:

1. Edit the setenv.cmd, located in `<beahome>\weblogic81\samples\LiquidData\EvalGuide`, to point to your WebLogic Server installation. This will set the environment for running clientgen. For example:

   ```
   call <beahome>\weblogic81\server\bin\setWLSEnv.cmd
   set CLASSPATH=d:\bea\weblogic81\server\lib\webservices.jar;%CLASSPATH%
   echo %CLASSPATH%
   ```

2. Open the command prompt.
3. Navigate to the `<beahome>\weblogic81\samples\LiquidData\EvalGuide` folder.
4. Run `setenv.cmd`.
5. Run clientgen.cmd to generate `CreditRatingWSClient.jar`.

6. In WebLogic Workshop add `CreditRatingWSClient.jar` to your application's Libraries folder. The .jar file should be located in `<beahome>\weblogic81\samples\LiquidData\EvalGuide`.

24.2 Writing Web Service Update Logic in the Update Override

You now should set the update override class to the CreditRatingExit. This will let you get any updated credit rating information, invoke the CreditRating Web service, and pass in the new value.

Objectives

In this exercise, you will:

- Import the `CreditRatingExit.java` file into the WebServices folder.
- Set the update override class to the CreditRatingExit.

Instructions

1. Right-click the WebServices folder, located in the DataServices folder.
2. Choose Import.
3. Navigate to `<beahome>\weblogic81\samples\LiquidData\EvalGuide` and select CreditRatingExit.java.
4. Click Import.
5. Build the DataServices project.
6. Open `getCreditRatingResponse.ds` in Design View. The file is located in the WebServices folder.
7. In the Property Editor, set the update override class by selecting CreditRatingExit from `DataServices\WebServices`.
8. Build the DataServices project.

24.3 Testing the Update Override

You are now ready to test whether the update override functions correctly.

Objectives

In this exercise, you will:

- Change a customer's credit rating.
- View the results.

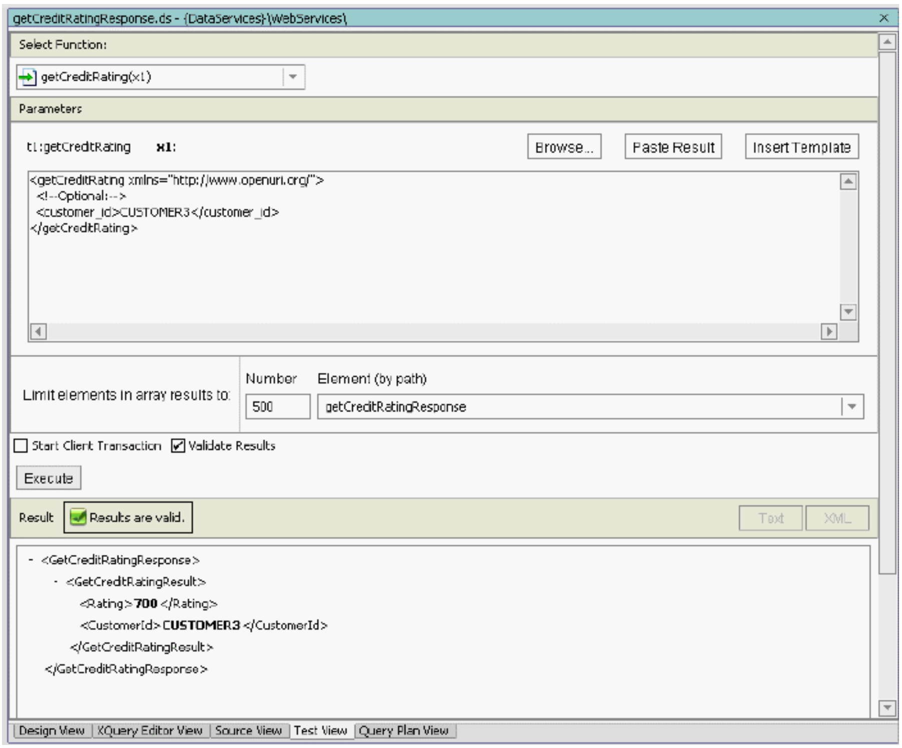
Instructions

1. Open `CreditRatingDBTest.jws`, located in the CreditRatingWS folder.
2. Click the Start icon. The Workshop Test Browser opens.
3. Enter CUSTOMER3 in the customer_id field and click `getCreditRating(x1).`
4. Click the Test XML tab.
5. Copy the SOAP body for the `getCreditRating()` function.

```
<getCreditRating xmlns="http://www.openuri.org/">
  <!--Optional:-->
  <customer_id>string</customer_id>
</getCreditRating>
```

6. Close the Workshop Test Browser.
7. Open `getCreditRatingResponse.ds` in Test View.
8. Paste the SOAP body into the Parameter field.
9. Change <customer_id>string</customer_id> to <customer_id>CUSTOMER3</customer_id>.
10. Click Execute.
11. Click Edit and modify the credit rating. The update override is functioning correctly if you can update the credit rating.

Figure 24-1 Test View of Update Override for a Web Service



Number: 500
Element (by path): getCreditRatingResponse

```
<getCreditRating xmlns="http://www.openuri.org/">
  <!--Optional:-->
  <customer_id>CUSTOMER3</customer_id>
</getCreditRating>
```

```
- <GetCreditRatingResponse>
    - <GetCreditRatingResult>
        <Rating> 700 </Rating>
        <CustomerId> CUSTOMER3 </CustomerId>
      </GetCreditRatingResult>
  </GetCreditRatingResponse>
```

24.4 Checking for Change Requirements

You can now use the Web service to perform update overrides.

Objectives

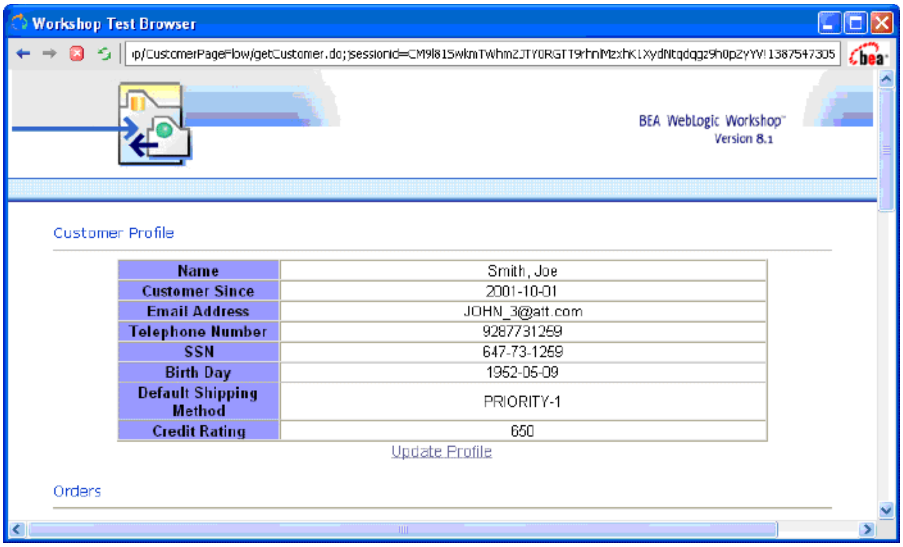
In this exercise you will:

- Change credit rating information from within your CustomerManagementWebApp application.
- Confirm update override results.

Instructions

1. Open `CustomerPageFlowController.jpf`, which is located in the CustomerManagementWebApp folder. The Workshop Test Browser opens.
2. Click the Start icon.
3. Enter CUSTOMER3 in the CUSTOMER ID field and click Submit.
4. Click Update Profile, change the credit rating information, click Submit, and then click Submit All Changes.
5. Confirm if the credit rating was updated, by clicking Back, entering CUSTOMER3 in the CUSTOMER ID field, and clicking Submit.

Figure 24-2 Workshop Test Browser View of Update Override Functionality


Workshop Test Browser
p/CustomerPageFlow/getCustomer.do;jsessionid=CM9l815wkmTWhmZJTY0RGTT9rhnMzxhK1XydNtqdqgz9h0pZyYV!1387547305
BEA WebLogic Workshop
Version 8.1
Customer Profile

Name	Smith, Joe
Customer Since	2001-10-01
Email Address	JOHN_3@att.com
Telephone Number	9287731259
SSN	647-73-1259
Birth Day	1952-05-09
Default Shipping Method	PRIORITY-1
Credit Rating	650

Lesson Summary

In this lesson, you learned how to:

- Create an update override for a physical data service (Web service)

- Associate the update override with a Web service client and write logic to invoke Web service update operations.
- Use the change summary to check whether there are changes needing to be written.

Overriding SQL Updates Using Update Overrides

So far you have completed a few lessons on how update override functionality can be used for custom data manipulation and web service updates.

In this lesson you will learn how custom SQL updates can be used for performing manual updates to a relational source (table, view, stored procedure, or SQL Exit), using update overrides and JDBC.

Objectives

After completing this lesson, you will be able to:

- Add update functionality to a previously created update override.
- Write an update override for performing manual updates to a relational source (table, view, stored procedure, or update override) via JDBC.
- Create an update override for a physical data service.
- Setup the update override to be a JDBC client and write logic to update the database table.

Overview

Update overrides are useful in situations where you need to perform some custom updates or create a custom query.

In this particular case, since the previous update override lacks update functionality, you can add an update statement to the override.

25.1 Adding SQL Update Statements to an Update Override File

You can add SQL update statements to an update override file, thereby enabling custom data manipulations in relational databases.

Objectives

In this exercise, you will:

- Import the Java folder, which contains the `MySQLExit.java` file.
- Add SQL update statements to the Java file.

Instructions

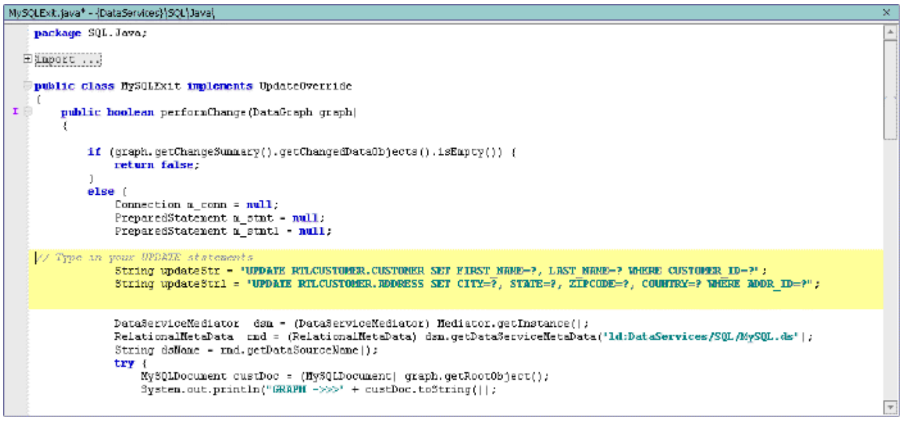
1. Right-click the SQL folder located in DataServices project, choose Import, and select the Java folder from the `<beahome>\weblogic81\samples\LiquidData\EvalGuide` folder.
2. Click Import and verify that the Java folder is added to the SQL folder.
3. Open `MySQLExit.java`, located in the `DataServices\SQL\Java` folder.
4. Locate the line "Type in your UPDATE SQL statements here".
5. Enter the two following SQL statements and store them into updateStr and updateStr1 respectively:

```
"UPDATE RTLCUSTOMER.CUSTOMER SET FIRST_NAME=?, LAST_NAME=? WHERE
CUSTOMER_ID=?";

"UPDATE RTLCUSTOMER.ADDRESS SET CITY=?, STATE=?, ZIPCODE=?, COUNTRY=?
WHERE ADDR_ID=?";
```

   Your code should look like the following:

```
String updateStr = "UPDATE RTLCUSTOMER.CUSTOMER SET FIRST_NAME=?,
LAST_NAME=? WHERE CUSTOMER_ID=?";

String updateStr1 = "UPDATE RTLCUSTOMER.ADDRESS SET CITY=?, STATE=?,
ZIPCODE=?, COUNTRY=? WHERE ADDR_ID=?";
```

Figure 25-1 MySQLExit.java



MySQLExit.java* - {DataServices}\SQL\Java\

```
package SQL.Java;

import ...

public class MySQLExit implements UpdateOverride
{
    public boolean performChange(DataGraph graph)
    {

        if (graph.getChangeSummary().getChangedDataObjects().isEmpty()) {
            return false;
        }
        else {
            Connection m_conn = null;
            PreparedStatement m_stmt = null;
            PreparedStatement m_stmt1 = null;

// Type in your UPDATE statements
            String updateStr = "UPDATE RTLCUSTOMER.CUSTOMER SET FIRST_NAME=?, LAST_NAME=? WHERE CUSTOMER_ID=?";
            String updateStr1 = "UPDATE RTLCUSTOMER.ADDRESS SET CITY=?, STATE=?, ZIPCODE=?, COUNTRY=? WHERE ADDR_ID=?";

            DataServiceMediator  dsm = (DataServiceMediator) Mediator.getInstance();
            RelationalMetaData  rmd = (RelationalMetaData) dsm.getDataServiceMetaData("ld:DataServices/SQL/MySQL.ds");
            String dsName = rmd.getDataSourceName();
            try {
                MySQLDocument custDoc = (MySQLDocument) graph.getRootObject();
                System.out.println("GRAPH ->>>" + custDoc.toString());
```

6. Save `MySQLExit.java` and close the file.

7. Build DataServices project.

25.2 Associating an SQL-Based Data Service and Update Override

You must now set the update override class to the MySQLExit. This will let you get any updated changes and pass the new value.

Objectives

In this exercise, you will:

- Associate the update override class with the MySQLExit.
- Confirm the settings in the Property Editor.

Instructions

1. Open `MySQL.ds` in Design View. The file is located in the `DataServices\SQL` folder.

2. Click the MySQL Data Service header. The Property Editor opens.

3. In the Property Editor, set the update override class by selecting MySQLExit from the `DataServices\SQL\Java` folder.

4. Save the `MySQL.ds` file.

5. Build your DataServices project.

25.3 Testing Updates

You are now ready to test whether the update override functions correctly.

Objectives

In this exercise, you will:

- Test the update override, by using the MySQL data service to make changes to the underlying relational data source.
- View the results.

Instructions

1. Open `MySQL.ds` in Test View.
2. Select `MySQL(x1)` from the Function drop-down list, enter CA, and click Execute.
3. Click Edit.
4. Test if updates are getting propagated to the database, by completing the following steps:
    a. Select any Customer node.
    b. Modify City and Zip Code elements.
    c. Click Submit to issue the update override commit command and propagate changes to the database.
5. Select `MySQL(x1)` from the function drop-down list, enter CA, and click Execute to confirm that your database is updated.

Lesson Summary

In this lesson, you learned how to:

- Create an update override for a physical data service.
- Setup the update override to be a JDBC client and write logic to update the database table.

Understanding Query Plans

A query plan contains detailed, functional-level information about an XQuery. Reviewing the Query Plan is the first step in troubleshooting a data service function's performance bottlenecks, as it lets you view the query's construction.

Objectives

After completing this lesson, you will be able to:

- Examine a query plan in three different views: tree, XML, and text.
- Locate the SQL statement created to retrieve data from the underlying database.
- Locate XML elements.

Overview

The most common reason for viewing a query plan is to review the SQL statement generated by the ALDSP query engine. However, the query plan also displays the following information for the physical data sources to be called during the query:

Physical Data Source	Information Provided
Relational	Data source name, actual SQL calls, and join parameters.
Web Services	Data source name, operation(s) called, and join parameters.

Physical Data Source	Information Provided
Custom Functions	Function name and join parameters.
XML and Delimited Files.	Filename

In addition, the following information is displayed for all functions:

- Number of invocations.
- Order in which the data source calls are made.
- Compilation time.
- Areas where calls are made in parallel.
- Areas where there are Cartesian joins.
- Areas where join algorithms are used, including parameter passing and index joins.
- Any calls to a middle-tier cache.

26.1 Viewing the Query Plan

A query plan is generated for each data service function, when a ALDSP project is built.

Objectives

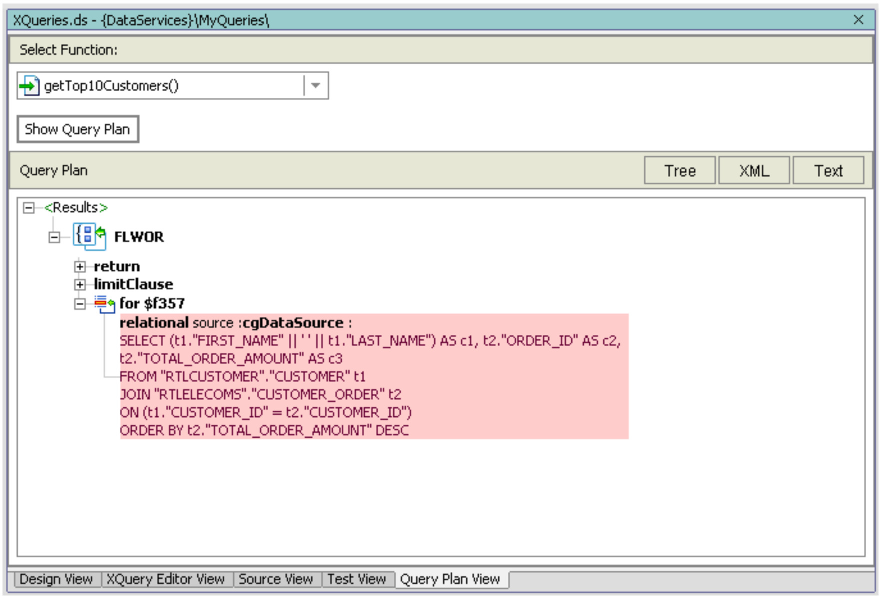
In this exercise, you will:

- Get the query plan for the `getCustomerProfile()` function.
- View the results in tree, XML, and text views.

Instructions

1. Open `XQueries.ds` in Query Plan View.
2. Select `getTop10Customers()` from the function drop down list.
3. Click Show Query Plan. The query plan opens in tree view, as displayed in Figure 26-1.

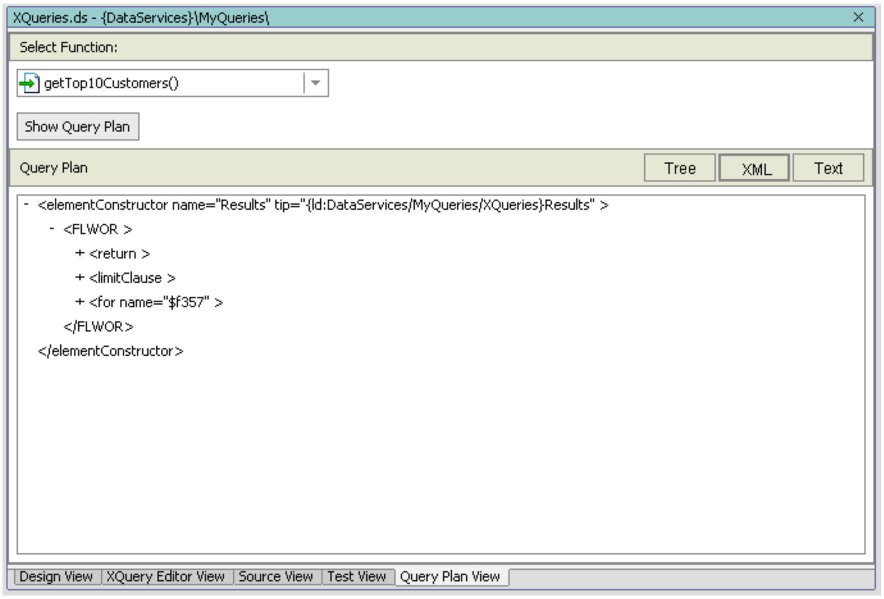
Figure 26-1 Query Plan as a Tree Structure



```
<Results>
  FLWOR
    return
    limitClause
    for $f357
      relational source :cgDataSource :
      SELECT (t1."FIRST_NAME" || ' ' || t1."LAST_NAME") AS c1, t2."ORDER_ID" AS c2,
      t2."TOTAL_ORDER_AMOUNT" AS c3
      FROM "RTLCUSTOMER"."CUSTOMER" t1
      JOIN "RTLELECOMS"."CUSTOMER_ORDER" t2
      ON (t1."CUSTOMER_ID" = t2."CUSTOMER_ID")
      ORDER BY t2."TOTAL_ORDER_AMOUNT" DESC
```

4. Click the XML button to view the Query Plan as an XML document.

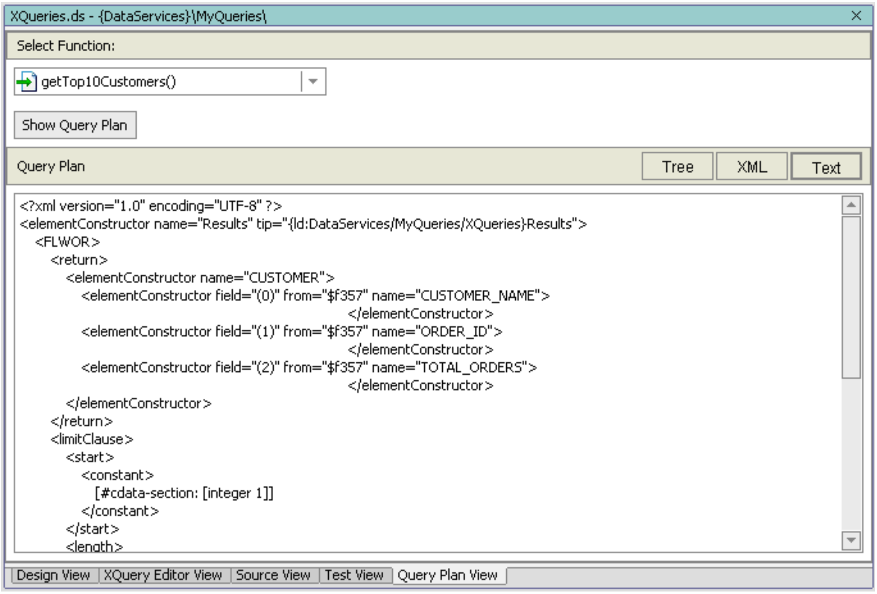
Figure 26-2 Query Plan as an XML Document



```
- <elementConstructor name="Results" tip="{ld:DataServices/MyQueries/XQueries}Results" >
    - <FLWOR >
        + <return >
        + <limitClause >
        + <for name="$f357" >
      </FLWOR>
  </elementConstructor>
```

5. Click the Text button to view the Query Plan as a text document.

Figure 26-3 Query Plan as a Text Document



```
<?xml version="1.0" encoding="UTF-8" ?>
<elementConstructor name="Results" tip="{ld:DataServices/MyQueries/XQueries}Results">
  <FLWOR>
    <return>
      <elementConstructor name="CUSTOMER">
        <elementConstructor field="(0)" from="$f357" name="CUSTOMER_NAME">
                                              </elementConstructor>
        <elementConstructor field="(1)" from="$f357" name="ORDER_ID">
                                              </elementConstructor>
        <elementConstructor field="(2)" from="$f357" name="TOTAL_ORDERS">
                                              </elementConstructor>
      </elementConstructor>
    </return>
    <limitClause>
      <start>
        <constant>
          [#cdata-section: [integer 1]]
        </constant>
      </start>
      <length>
```

26.2 Locating the SQL Statement in a Query Plan

SQL statements are generated for functions that call relational databases.

Objectives

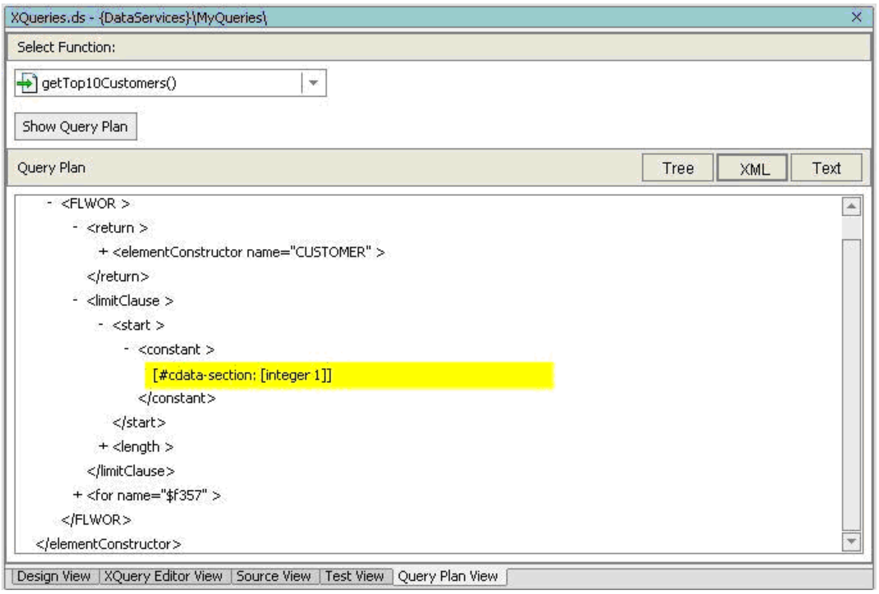
In this exercise, you will:

- Locate an SQL statement within the query.
- Review the contents of the SQL statement.

Instructions

1. Open the Query Plan as an XML document.
2. Expand the FLWOR nodes until you see the #cdata-section. This is the SQL statement for the query.

Figure 26-4 Query Plan View of SQL Statements



```
  - <FLWOR >
    - <return >
      + <elementConstructor name="CUSTOMER" >
      </return>
    - <limitClause >
      - <start >
        - <constant >
            [#cdata-section: [integer 1]]
          </constant>
        </start>
      + <length >
      </limitClause>
    + <for name="$f357" >
  </FLWOR>
</elementConstructor>
```

As a reminder, this function retrieves customer and order amount information. In addition, the result set is ordered in descending order by order amount.

26.3 Locating XML Elements

XML elements identify the data that will be returned by the query function. Each XML element is identified with a QName.

Objectives

In this exercise, you will:

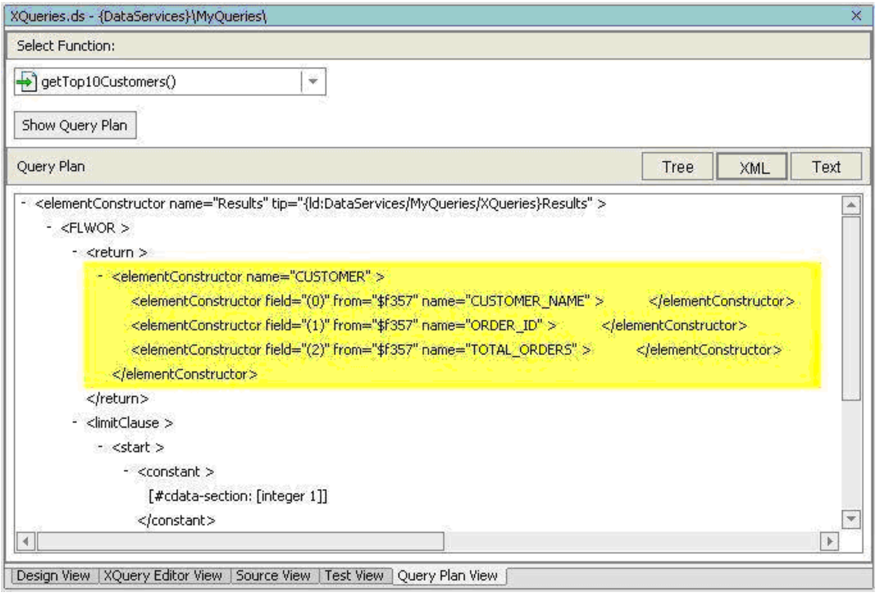
- Locate all XML elements within the query.
- Review the contents of the XML element lines.

Instructions

1. In Query Plan View, expand the return node.

2. Notice all the XML elements that will be returned when the function is executed.

Figure 26-5 Query Plan View of XML Elements



```
- <elementConstructor name="Results" tip="{ld:DataServices/MyQueries/XQueries}Results" >
    - <FLWOR >
        - <return >
            - <elementConstructor name="CUSTOMER" >
                <elementConstructor field="(0)" from="$f357" name="CUSTOMER_NAME" >     </elementConstructor>
                <elementConstructor field="(1)" from="$f357" name="ORDER_ID" >     </elementConstructor>
                <elementConstructor field="(2)" from="$f357" name="TOTAL_ORDERS" >     </elementConstructor>
              </elementConstructor>
          </return>
        - <limitClause >
            - <start >
                - <constant >
                    [#cdata-section: [integer 1]]
                  </constant>
```

Lesson Summary

In this lesson, you learned how to:

- Examine a query plan as tree, XML, and text documents.
- Locate the SQL statement that was created to retrieve data from the underlying database.
- Locate XML elements.

Reusing XQuery Code through Vertical View Unfolding

ALDSP enables powerful data service code reusability.

Objectives

After completing this lesson, you will be able to:

- Re-use code.
- Unfold vertical file view.

Overview

ALDSP enables powerful data service code reusability. You can develop your logic once, and then re-use it later when building other data services. This feature is called view unfolding.

In addition to code reuse, ALDSP is smart enough to optimize your output and only query sources and elements that you request in your data service (vertical view unfolding).

27.1 Unfolding Vertical View

You will reuse the CustomerProfile data service previously built to retrieve Customer Order information. The CustomerProfile data service is built from three different tables in the underlying PointBase database: CUSTOMER, CUSTOMER_ORDER and CUSTOMER_ORDER_LINE_ITEM.

Objectives

In this exercise, you will:

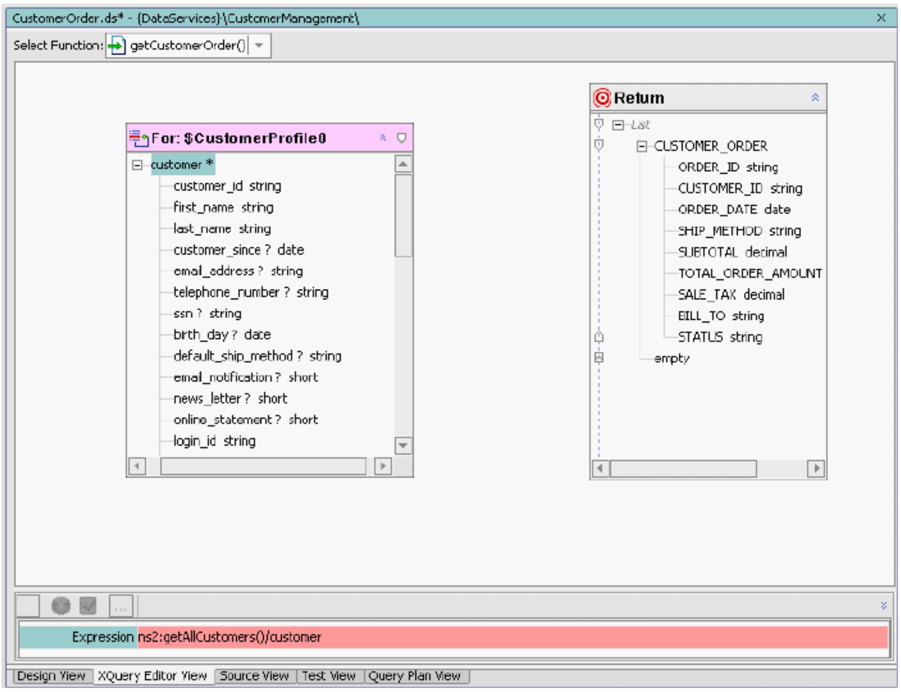
- Import the CustomerOrder data service into the CustomerManagement folder.
- Import `CustomerOrder.xsd`, and then associate the schema with the CustomerOrder data service.
- Implement a query function, and define its conditions.

Instructions

1. Import `CustomerOrder.ds` into `DataServices\CustomerManagement`. The file is located in `<beahome>\weblogic81\samples\LiquidData\EvalGuide`.
2. Import `CustomerOrder.xsd` into `DataServices\CustomerManagement\schemas`. The file is also located in `<beahome>\weblogic81\samples\LiquidData\EvalGuide`.
3. Implement the `getCustomerOrder()` function in the CustomerOrder data service, by completing the following steps:
   a. Open `CustomerOrder.ds` in XQuery Editor View.
   b. In Data Services Palette, drag and drop `getAllCustomers()` into XQuery Editor View. The method call is located in the folder:

      ```
      DataServices\CustomerManagement\CustomerProfile
      ```
4. Set the conditions for the function, by completing the following steps:
   a. Select the Customer* element. This will activate the Expression Editor and make visible the `ns2:getAllCustomers()/customer` expression. You will use the Expression Editor to scope the data returned in the `getAllCustomers()` function.

Figure 27-1 Default Expression



Expression ns2:getAllCustomers()/customer

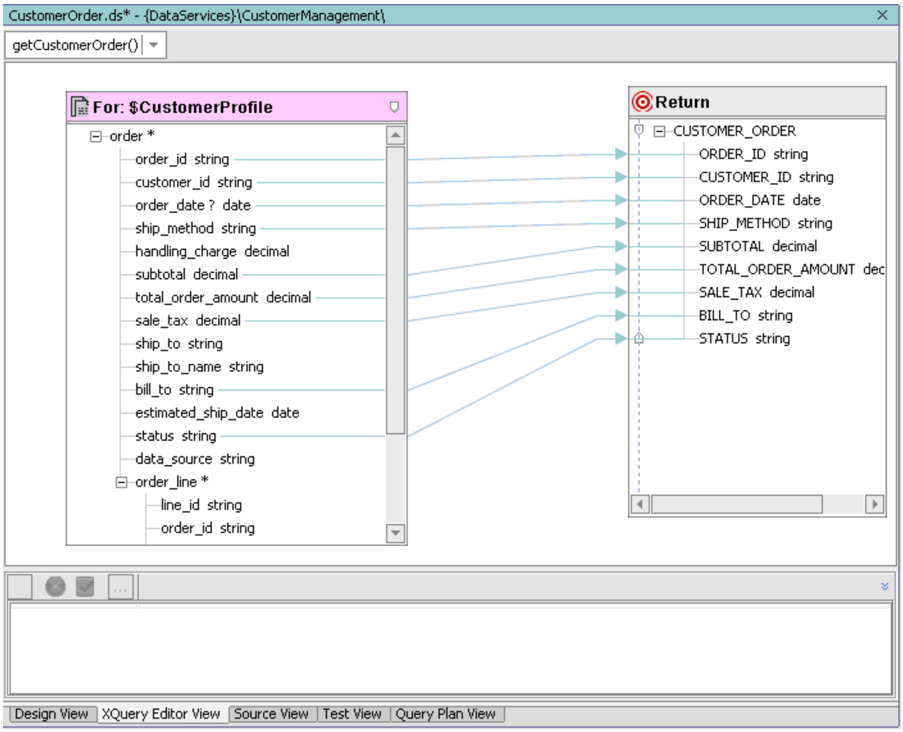
b. Triple-click the Expression field.

c. Modify the expression by adding the following code:

```
ns2:getAllCustomers()/customer/orders/order
```

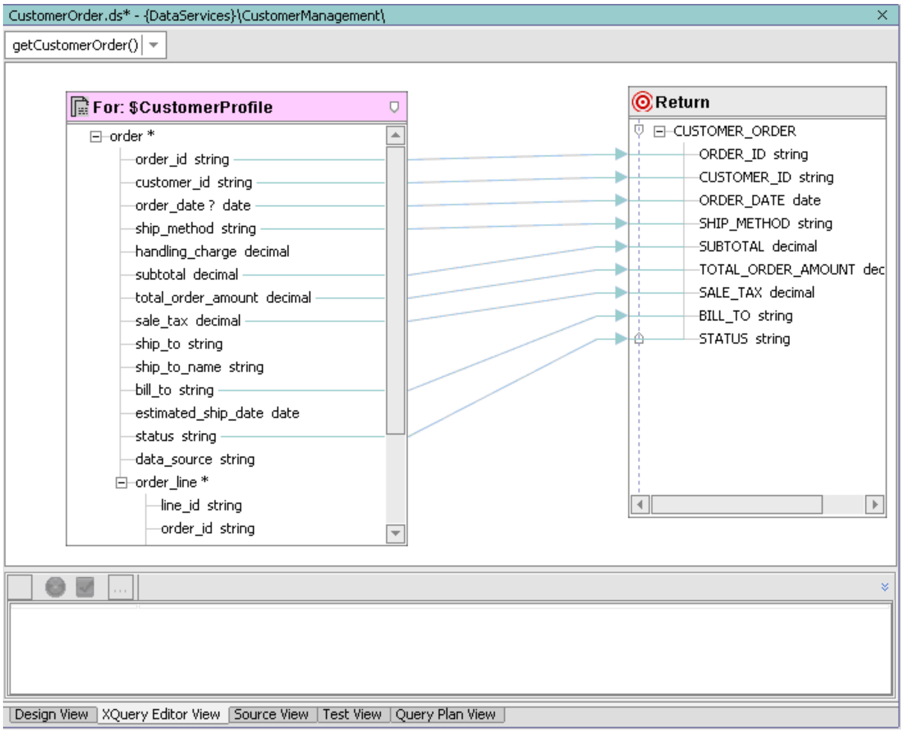
d. Click the green checkmark icon to accept the changes. The CustomerProfile* element changes to the order* element, and the For:$CustomerProfile schema now includes the order elements.

Figure 27-2 Modified Expression



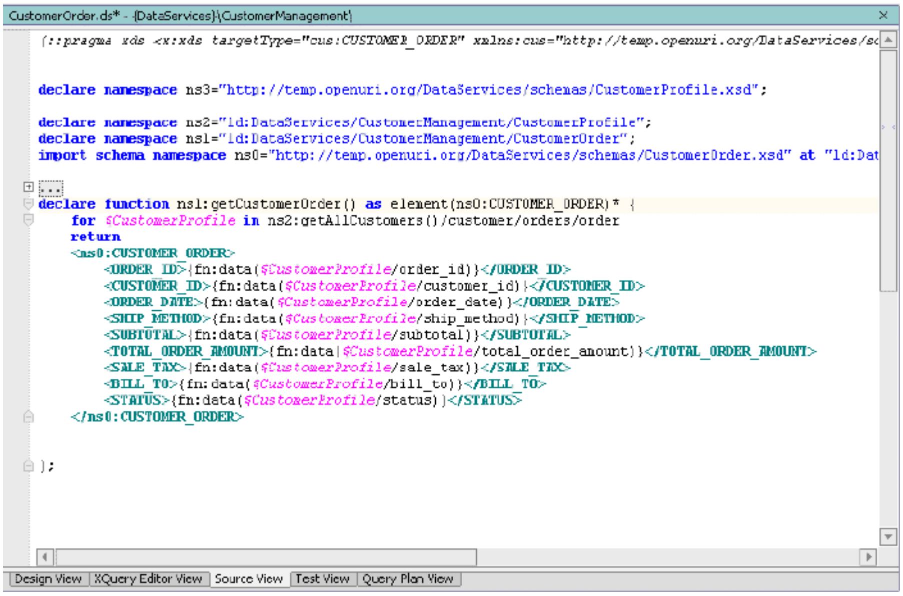
5. Create a simple mapping: Drag and drop all order* elements (source node) to the corresponding CUSTOMER_ORDER elements in the Return type.

Figure 27-3 XQuery Editor View—Mappings



6. Save the data service file.
7. Open `CustomerOrders.ds` in Source View and notice that the function is using the CustomerProfile file as its data source.

Figure 27-4 Source View of Vertical File Unfolding Function



```
(::pragma xds <x:xds targetType="cus:CUSTOMER_ORDER" xmlns:cus="http://temp.openuri.org/DataServices/sc


declare namespace ns3="http://temp.openuri.org/DataServices/schemas/CustomerProfile.xsd";

declare namespace ns2="ld:DataServices/CustomerManagement/CustomerProfile";
declare namespace ns1="ld:DataServices/CustomerManagement/CustomerOrder";
import schema namespace ns0="http://temp.openuri.org/DataServices/schemas/CustomerOrder.xsd" at "ld:Dat

...
declare function ns1:getCustomerOrder() as element(ns0:CUSTOMER_ORDER)* {
    for $CustomerProfile in ns2:getAllCustomers()/customer/orders/order
    return
    <ns0:CUSTOMER_ORDER>
        <ORDER_ID>{fn:data($CustomerProfile/order_id)}</ORDER_ID>
        <CUSTOMER_ID>{fn:data($CustomerProfile/customer_id)}</CUSTOMER_ID>
        <ORDER_DATE>{fn:data($CustomerProfile/order_date)}</ORDER_DATE>
        <SHIP_METHOD>{fn:data($CustomerProfile/ship_method)}</SHIP_METHOD>
        <SUBTOTAL>{fn:data($CustomerProfile/subtotal)}</SUBTOTAL>
        <TOTAL_ORDER_AMOUNT>{fn:data($CustomerProfile/total_order_amount)}</TOTAL_ORDER_AMOUNT>
        <SALE_TAX>{fn:data($CustomerProfile/sale_tax)}</SALE_TAX>
        <BILL_TO>{fn:data($CustomerProfile/bill_to)}</BILL_TO>
        <STATUS>{fn:data($CustomerProfile/status)}</STATUS>
    </ns0:CUSTOMER_ORDER>


};
```

27.2 Testing a Vertical File Unfolding

Testing a vertical file unfolding is similar to testing any other data service function.

Objectives

In this exercise, you will:

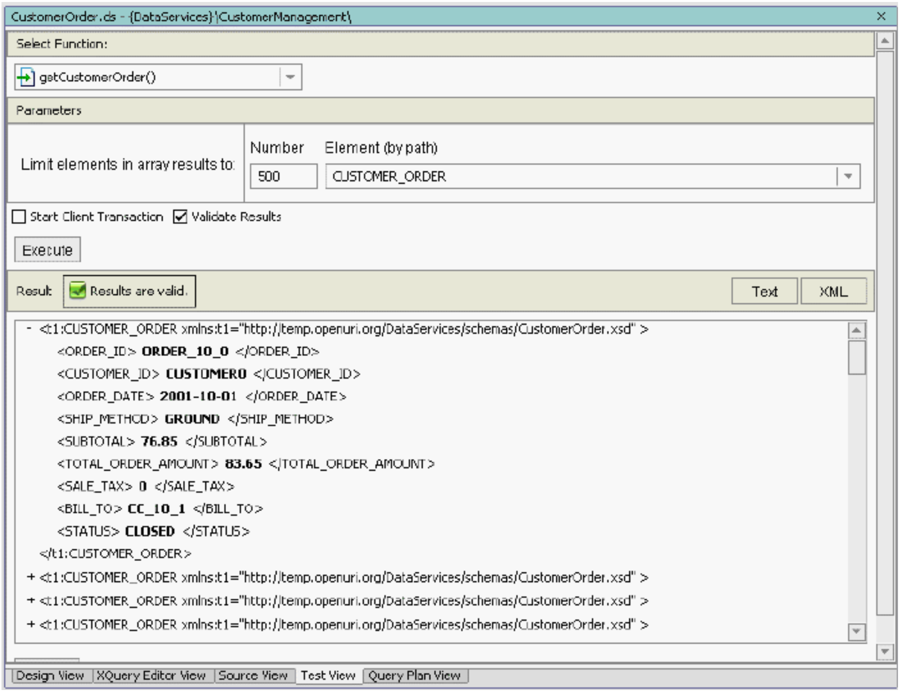
- Test the CustomerOrder data service.
- Review the results.

Instructions

1. Open `CustomerOrders.ds` in Test View.
2. Select `getCustomerOrder()` from the function drop-down list.
3. Click Execute.

4. Confirm that you can retrieve customer order information.

Figure 27-5 Vertical File Unfolding Test Results



```
- <t1:CUSTOMER_ORDER xmlns:t1="http://temp.openuri.org/DataServices/schemas/CustomerOrder.xsd" >
    <ORDER_ID> ORDER_10_0 </ORDER_ID>
    <CUSTOMER_ID> CUSTOMER0 </CUSTOMER_ID>
    <ORDER_DATE> 2001-10-01 </ORDER_DATE>
    <SHIP_METHOD> GROUND </SHIP_METHOD>
    <SUBTOTAL> 76.85 </SUBTOTAL>
    <TOTAL_ORDER_AMOUNT> 83.65 </TOTAL_ORDER_AMOUNT>
    <SALE_TAX> 0 </SALE_TAX>
    <BILL_TO> CC_10_1 </BILL_TO>
    <STATUS> CLOSED </STATUS>
  </t1:CUSTOMER_ORDER>
+ <t1:CUSTOMER_ORDER xmlns:t1="http://temp.openuri.org/DataServices/schemas/CustomerOrder.xsd" >
+ <t1:CUSTOMER_ORDER xmlns:t1="http://temp.openuri.org/DataServices/schemas/CustomerOrder.xsd" >
+ <t1:CUSTOMER_ORDER xmlns:t1="http://temp.openuri.org/DataServices/schemas/CustomerOrder.xsd" >
```

Lesson Summary

In this lesson, you learned how to:

- Build a data service based on another data service (view unfolding)
- Re-use code (vertical file unfolding).

Configuring Alternatives for Unavailable Data Sources

Sometimes a particular data source is either temporarily unavailable or very slow to send a response back to a consuming application. In such cases, you need to be able to run an alternative data source. ALDSP enables you create an alternative data source that will be called if the primary data source does not respond within a specified time frame.

Objectives

After completing this lesson, you will be able to:

- Invoke, configure, and test an alternative data source.
- Use the `fn-bea:timeout()` function for configuring alternative sources.
- Review WebLogic Server output.

Overview

Enabling an alternative data source is implemented by calling the `fn-bea:timeout()` function. The syntax for the function is as follows:

```
fn-bea:timeout($seq as item()*, $millis as xs:int, $alt as item()*) as
item()*
```

where:

- $seq is the primary expression.

- $millis in the timeout in milliseconds.
- $alt is the alternate expression.

To implement this functionality, the return types of both the primary and alternative expression should be available when the project is compiled. This ensures that the function's return type is correctly inferred. In other words, the source metadata must be available at compile time, because the alternative source function provides only runtime failover capability.

28.1 Setting the Demonstration Conditions

You will import a slow Web service into your application, thereby enabling the demonstration of configuring alternatives for unavailable data sources.

Objectives

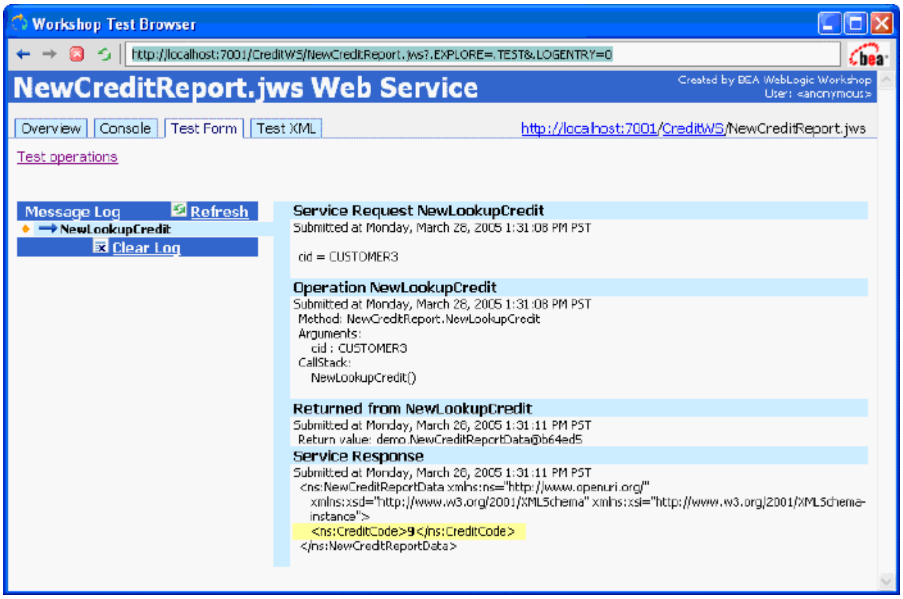
In this exercise, you will:

- Import and test a "slow" Web for demonstration purposes.
- Create a physical data service that is based on an alternative data source.

Instructions

1. Right-click the Evaluation folder and then import the `<beahome>\weblogic81\samples\LiquidData\EvalGuide\CreditWS` file as a Web Service Project. This will import a simple Web service that does nothing but sleep for 3 seconds. Click 'Yes' when asked for "Files required for Web Services are not in the project. Do you wish to add them?"
2. Build the CreditWS project.
3. Test the slow Web service by completing the following steps:
   a. Open the `NewCreditReport.jws`, located in the CreditWS folder.
   b. Click the Start icon (or press Ctrl + F5). The Workshop test browser opens.
   c. Enter CUSTOMER3 in the cid field and click NewLookupCredit.
   d. Confirm that you can get credit rating information.

Figure 28-1 Test Browser View of the Slow Web Service



NewCreditReport.jws Web Service

Created by BEA WebLogic Workshop
User: <anonymous>

http://localhost:7001/CreditWS/NewCreditReport.jws

Service Request NewLookupCredit
```
Submitted at Monday, March 28, 2005 1:31:08 PM PST

 cid = CUSTOMER3
```

Operation NewLookupCredit
```
Submitted at Monday, March 28, 2005 1:31:08 PM PST
 Method: NewCreditReport.NewLookupCredit
 Arguments:
    cid : CUSTOMER3
 CallStack:
    NewLookupCredit()
```

Returned from NewLookupCredit
```
Submitted at Monday, March 28, 2005 1:31:11 PM PST
 Return value: demo.NewCreditReportData@b64ed5
```

Service Response
```
Submitted at Monday, March 28, 2005 1:31:11 PM PST
 <ns:NewCreditReportData xmlns:ns="http://www.openuri.org/"
   xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns:xsi="http://www.w3.org/2001/XMLSchema-
   instance">
   <ns:CreditCode>9</ns:CreditCode>
 </ns:NewCreditReportData>
```

4. Create a physical data service for the slow Web service, by completing the following steps:
   a. Select the Overview tab in the Workshop Test Browser.
   b. Click Complete WSDL.
   c. Copy the WSDL URI, which you will use to import an alternative data service. The URI typically is:

      `http://localhost:7001/CreditWS/NewCreditReport.jws?WSDL=`

   d. In the Application pane of WebLogic Workshop, right-click the WebServices folder (located in DataServices).
   e. Choose Import Source Metadata.
   f. Select Web Service from the Data Source Type drop-down list and click Next.
   g. Paste the WSDL URI into the URI field, then click Next.
   h. Expand the folders and select the NewLookupCredit operation.
   i. Click Add to populate the Selected Web Service Operations pane and click Next.

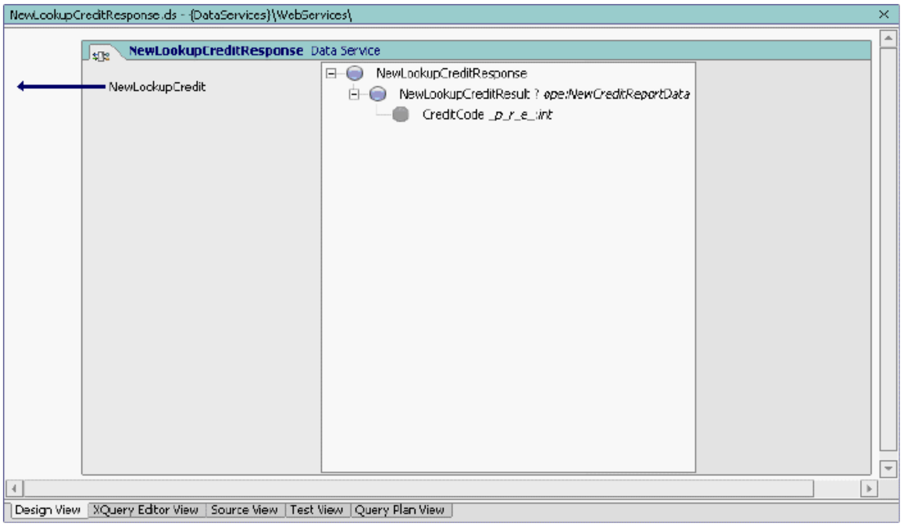
Note: Do not select NewLookCredit as a side-effect procedure.

j. Review the Summary information and click Finish.

5. Check the Application pane. There should be a new physical data service called `NewLookupCreditResponse.ds`.

6. Open `NewLookupCreditResponse.ds` in Design View. There should be a function called NewLookupCredit.

Figure 28-2 Design View of Web Service-Based Data Service


NewLookupCreditResponse.ds - {DataServices}\WebServices\
NewLookupCreditResponse Data Service
NewLookupCredit
NewLookupCreditResponse
NewLookupCreditResult ? ope:NewCreditReportData
CreditCode _p_r_e_:int
Design View
XQuery Editor View
Source View
Test View
Query Plan View

28.2 Configuring Alternative Sources

Because the CreditWS Web service is slow, you need to configure an alternative source to obtain the credit rating information in a timely manner.

Objectives

In this exercise, you will:

- Configure an alternative data source.
- Use the `fn:bea:timeout()` function.

Instructions

1. Open `CustomerProfile.ds` in Source View. (The file is located in `DataServices\CustomerManagement`.
2. Add the following code to the namespace declaration:

   ```
   declare namespace
   ws3="ld:DataServices/WebServices/NewLookupCreditResponse";

   declare namespace ws4 = "http://www.openuri.org/";
   ```

3. Locate the `getAllCustomers()` function.
4. Locate the following entry:

   ```
   {

   for $rating  in ws1:getCreditRating(

       <ws2:getCreditRating>

           <ws2:customer_id>{data($CUSTOMER/CUSTOMER_ID)}</ws2:customer_id>

       </ws2:getCreditRating> )

   return

       <creditrating>


   <rating>{data($rating/ws2:getCreditRatingResult/ws2:Rating)}</rating>


   <customer_id>{data($rating/ws2:getCreditRatingResult/ws2:Customer_id)}<
   /customer_id>

       </creditrating>

   }
   ```

5. Replace that entry with the following code and note the use of the `fn-bea:timeout()` function.:

   ```
   {

   <creditrating>

     <rating>

       {

       fn-bea:timeout(

           data(
   ```

```
            ws3:NewLookupCredit(
                <ws4:NewLookupCredit>
                    <ws4:cid>{data($CUSTOMER/CUSTOMER_ID)}</ws4:cid>
                </ws4:NewLookupCredit>
                )/ws4:NewLookupCreditResult/ws4:CreditCode
        )
        , 2000,
        data(
            ws1:getCreditRating(
                <ws2:getCreditRating>

<ws2:customer_id>{data($CUSTOMER/CUSTOMER_ID)}</ws2:customer_id>
                </ws2:getCreditRating>
)/ws2:getCreditRatingResult/ws2:Rating)
        )
    }
  </rating>
  <customer_id>{data($CUSTOMER/CUSTOMER_ID)}</customer_id>
</creditrating>
}
```

28.3 Testing an Alternative Source

Testing `getAllCustomers()` function will let you confirm that the query is retrieving data from the alternative source, rather than the CreditWS.

Objectives

In this exercise, you will:

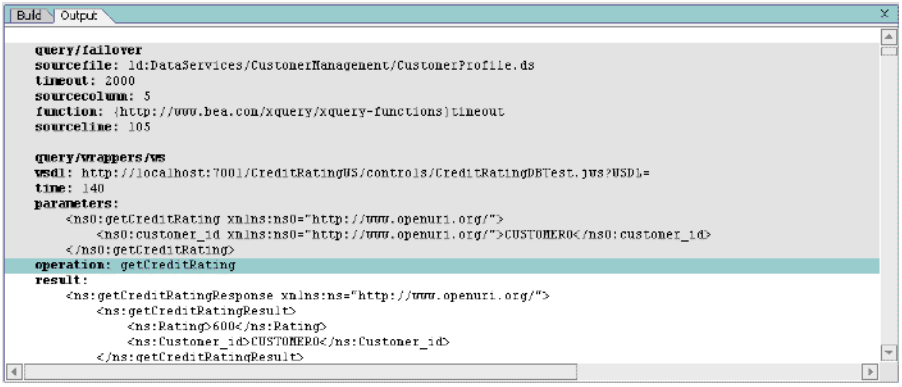
- Test the CustomerProfile data service, using the `getAllCustomers()` function.
- Review the results in the Output window.

Instructions

1. Build the DataServices project.
2. Enable auditing for the `getAllCustomers()` function using the AquaLogic Data Services Platform Console. For details about auditing, refer to `http://edocs.bea.com/aldsp/docs21/admin/monitor.html`.
3. Open `CustomerProfile.ds` located in CustomerManagement folder in Test View.
4. Select `getAllCustomers()` from the function drop-down list.
5. Click Execute.

   Open the Output window, scroll to the bottom, and then confirm that the CreditWS Web service times out and then the CreditRating Web service is called. The output should display as shown in Figure 28-3.

Figure 28-3 Output Window



```
    query/failover
    sourcefile: ld:DataServices/CustomerManagement/CustomerProfile.ds
    timeout: 2000
    sourcecolumn: 5
    function: {http://www.bea.com/xquery/xquery-functions}timeout
    sourceline: 105

    query/wrappers/ws
    wsdl: http://localhost:7001/CreditRatingWS/controls/CreditRatingDBTest.jws?WSDL=
    time: 140
    parameters:
        <ns0:getCreditRating xmlns:ns0="http://www.openuri.org/">
            <ns0:customer_id xmlns:ns0="http://www.openuri.org/">CUSTOMER0</ns0:customer_id>
        </ns0:getCreditRating>
    operation: getCreditRating
    result:
        <ns:getCreditRatingResponse xmlns:ns="http://www.openuri.org/">
            <ns:getCreditRatingResult>
                <ns:Rating>600</ns:Rating>
                <ns:Customer_id>CUSTOMER0</ns:Customer_id>
            </ns:getCreditRatingResult>
```

The invocation of the first Web service NewLookupCreditResponse fails because the thread times out. Because this Web service has failed it will not be invoked again. Instead, the alternate Web service is invoked.

Lesson Summary

In this lesson, you learned how to:

- Invoke, configure, and test an alternative data source.

- Use the `fn-bea:timeout()` function for configuring alternative sources.
- Review WebLogic Server output.

Enabling Fine Grained Caching

Fine-grained caching lets you cache a data subset, such as information that does not frequently change. Fine-grained caching is at the function level, because a function's role is to retrieve specific information.

Objectives

After completing this lesson, you will be able to:

- Define a cache policy for the slow credit rating Web service.
- Testing caching performance.

Overview

ALDSP provides a flexible caching mechanism to manage caching of data service functions. In Part 1, you learned how to cache a function in a logical data service. However, there are situations where you may want to cache only a sub-set of information available in a particular logical data service. For example, the CustomerProfile data service includes information about each customer's profile and order information. The profile information does not change often, whereas order information constantly changes. In this situation users would like to cache the profile information for a given customer but retrieve the most recent order information from the operational system.

By defining different caching policies for the underlying customer and order physical data services, you can cache only the CUSTOMER physical data service. As a result, any request made to the logical

CustomerProfile data service will be partly answered from the ALDSP Cache for customer information and partly answered from the operational system for order information.

29.1 Enabling Function-Level Caching for a Physical Data Service

Caching of a function in an underlying data service provides you with the ability to cache a sub-set of data within a data service function.

Objectives

In this exercise, you will:

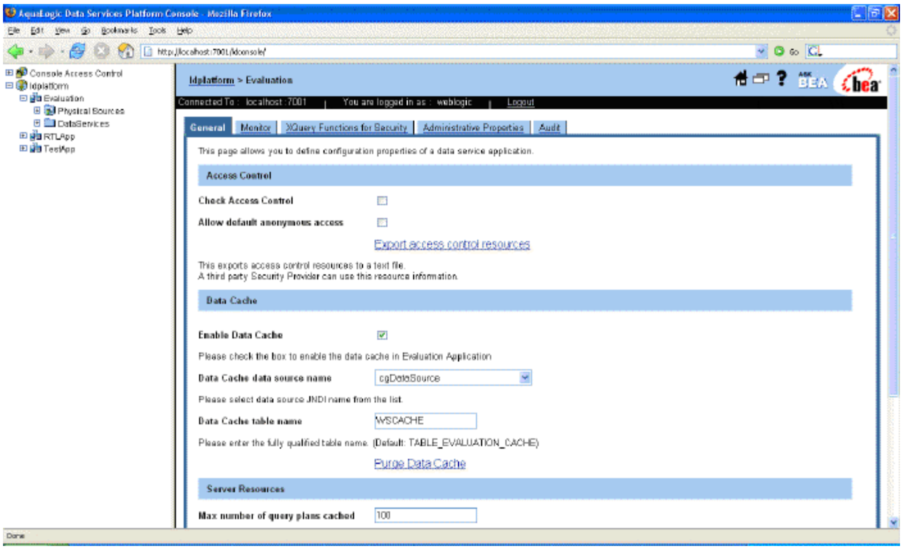
- Enable application-level caching and function-level caching.

Instructions

1. Login to the ALDSP Console (`http://localhost:7001/ldconsole/`), using the following credentials:
   - User Name = weblogic
   - Password = weblogic
2. Using the + icon, expand the ldplatform directory.

   Note: If you click the ldplatform name, the Application List page opens. This is not the page you want for this lesson.
3. Click Evaluation. The Administration Control's General page opens.
4. In the Data Cache section, select Enable Data Cache.
5. Select cgDataSource from the Data Cache data source name drop-down list.
6. Enter WSCACHE in the Data Cache table name field.
7. Click Apply.

Figure 29-1 Enable Caching


AquaLogic Data Services Platform Console - Mozilla Firefox
File Edit View Go Bookmarks Tools Help
http://localhost:7001/ldconsole/
Console Access Control
ldplatform
Evaluation
Physical Sources
DataServices
RTLApp
TestApp
ldplatform > Evaluation
Connected To : localhost :7001 | You are logged in as : weblogic | Logout
General
Monitor
XQuery Functions for Security
Administrative Properties
Audit
Done

This page allows you to define configuration properties of a data service application.

Access Control

Check Access Control ☐

Allow default anonymous access ☐

Export access control resources

This exports access control resources to a text file.
A third party Security Provider can use this resource information.

Data Cache

Enable Data Cache ✓

Please check the box to enable the data cache in Evaluation Application

Data Cache data source name: cgDataSource

Please select data source JNDI name from the list.

Data Cache table name: WSCACHE

Please enter the fully qualified table name. (Default: TABLE_EVALUATION_CACHE)

Purge Data Cache

Server Resources

Max number of query plans cached: 100

8. Expand the Evaluation folder and navigate to `getCreditRatingResponse.ds`, located in `DataServices\WebServices` folder.

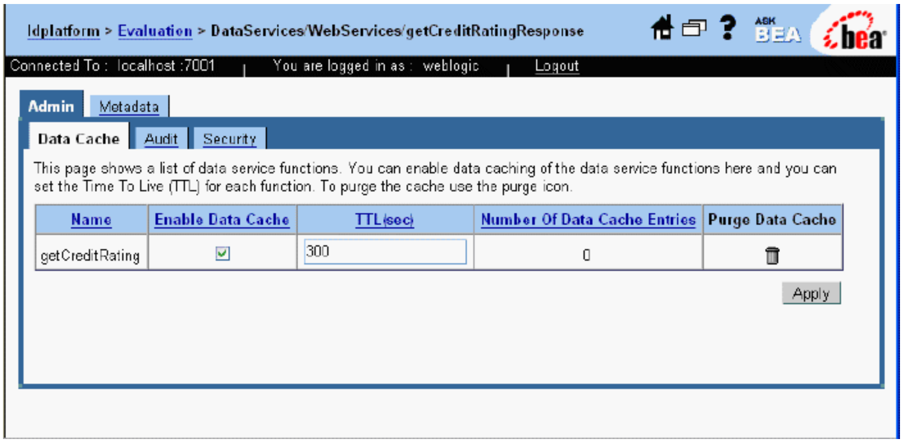
9. For the `getCreditRating()` function, set a caching policy by completing the following steps:

   a. Select Enable Data Cache.

   b. Enter 300 in the TTL field.

   c. Click Apply.

Figure 29-2 Enable Function-Level Caching


ldplatform > Evaluation > DataServices/WebServices/getCreditRatingResponse
Connected To : localhost :7001
You are logged in as : weblogic
Logout
Admin
Metadata
Data Cache
Audit
Security
Apply

This page shows a list of data service functions. You can enable data caching of the data service functions here and you can set the Time To Live (TTL) for each function. To purge the cache use the purge icon.

Name	Enable Data Cache	TTL(sec)	Number Of Data Cache Entries	Purge Data Cache
getCreditRating	✓	300	0	🗑

29.2 Testing the Caching Policy

You are now ready to test your new fine-grained caching policy.

Objectives

In this exercise, you will:

- Test the function-level caching policy.
- Determine whether the cache was populated.

Instructions

1. In WebLogic Workshop, execute a test query by completing the following steps:
   a. Open `CustomerProfile.ds` in Test View. The file is located in the CustomerManagement folder.
   b. Select `getCustomerProfile(CustomerID)` from the function drop-down list.
   c. Enter CUSTOMER3 in the Parameter field.
   d. Click Execute.

e. In the Output window, note the number of invocations and the times for the NewLookupCreditResponse and getCreditRatingResponse data sources.

2. In the PointBase Console, check whether the cache database table was populated by completing the following steps:

   a. Start the PointBase Console, using the following command in a command prompt window:

   ```
   <beahome>\weblogic81\common\bin\startPointBaseConsole.cmd
   ```

   b. Use the following configuration to connect to your local PointBase database:

   – Driver: com.pointbase.jdbc.jdbcUniversalDriver

   – URL: jdbc:pointbase:server://localhost:9093/workshop

   – User: weblogic

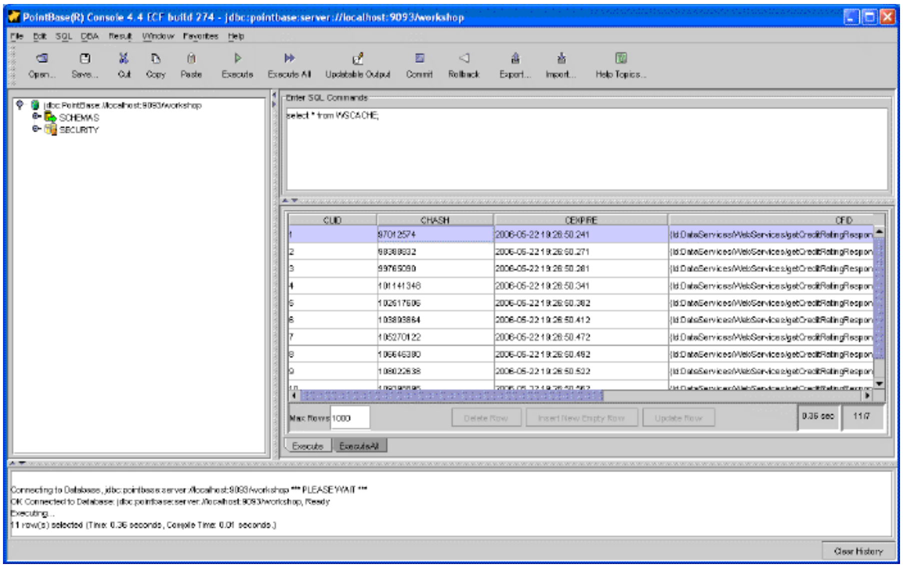
   – Password: weblogic

   c. Click OK.

   d. Enter the SQL command: SELECT * FROM WSCACHE

   e. Click Execute to check whether the cache was populated.

Figure 29-3 PointBase Console Cache Information


PointBase(R) Console 4.4 ECF build 274 - jdbc:pointbase:server://localhost:9093/workshop
File Edit SQL DBA Result Window Favorites Help
Open... Save... Cut Copy Paste Execute Execute All Updatable Output Commit Rollback Export... Import... Help Topics...
jdbc:PointBase://localhost:9093/workshop
SCHEMAS
SECURITY
Enter SQL Commands
Delete Row
Insert New Empty Row
Update Row
Execute
ExecuteAll
Clear History

```
select * from WSCACHE;
```

CUID	CHASH	CEXPIRE	CFID
1	97012574	2006-05-22 19:26:50.241	{ld:DataServices/WebServices/getCreditRatingRespon
2	98388832	2006-05-22 19:26:50.271	{ld:DataServices/WebServices/getCreditRatingRespon
3	99765090	2006-05-22 19:26:50.281	{ld:DataServices/WebServices/getCreditRatingRespon
4	101141348	2006-05-22 19:26:50.341	{ld:DataServices/WebServices/getCreditRatingRespon
5	102517606	2006-05-22 19:26:50.382	{ld:DataServices/WebServices/getCreditRatingRespon
6	103893864	2006-05-22 19:26:50.412	{ld:DataServices/WebServices/getCreditRatingRespon
7	105270122	2006-05-22 19:26:50.472	{ld:DataServices/WebServices/getCreditRatingRespon
8	106646380	2006-05-22 19:26:50.492	{ld:DataServices/WebServices/getCreditRatingRespon
9	108022638	2006-05-22 19:26:50.522	{ld:DataServices/WebServices/getCreditRatingRespon

Max Rows 1000 | 0.35 sec | 11/7

```
Connecting to Database, jdbc:pointbase:server://localhost:9093/workshop *** PLEASE WAIT ***
OK Connected to Database: jdbc:pointbase:server://localhost:9093/workshop, Ready
Executing...
11 row(s) selected (Time: 0.36 seconds, Compile Time: 0.01 seconds.)
```

29.3 Testing Performance Impact

The next step is to determine whether the caching policy improves query performance.

Objectives

In this exercise, you will:

- Execute a data service test.
- Determine whether the caching policy improved query performance time.

Instructions

1. In WebLogic Workshop, execute a test query by completing the following steps:
   a. Open `CustomerProfile.ds` in Test View. (The file is located in the CustomerManagement folder.)
   b. Select `getCustomerProfile()` from the function drop-down list.
   c. Enter CUSTOMER3 in the Parameter field.
   d. Click Execute.
2. Confirm the following performance results in the Output window:
   a. Confirm that the slow Web service (NewLookupCreditRatingResponse) was never invoked due to alternate path execution.
   b. Determine whether caching the Web service helped to reduce the query execution time.

Lesson Summary

In this lesson, you learned how to:

- Enable the cache for a physical data service function and define the cache's TTL.
- Determine the performance impact of the physical data service cache on a function in a logical data service by checking the query response time and whether the physical data service (original data source) was invoked.

CreatingXQueryFilters to Implement Conditional Logic Security

Data Services Platform can enable security based on the results of conditional logic.

Objectives

After completing this lesson, you will be able to:

- Activate security XQuery functions.
- Write security XQuery functions.

Overview

Conditional logic can be used to establish very specific security restrictions. For example, access to a social security number can be restricted to managers, as is illustrated in Exercise 30.2 Writing the XQuery Security Function. Security restrictions at the element level are set through the ALDSP Console.

30.1 Creating User Groups

The first step in setting conditional-logic security is establishing security groups.

Objectives

In this exercise, you will:

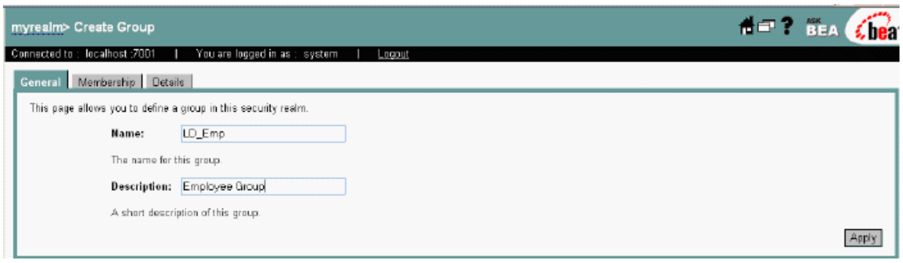
- Create new user groups.

- Assign user accounts to user groups.

Instructions

1. Login to the WebLogic Server Console (`http://localhost:7001/console/`), using the following credentials:
   - User Name = weblogic
   - Password = weblogic
2. Create two new user groups by completing the following steps:
   a. Choose Security → Realms → myrealm → Groups.
   b. Select Configure a New Group.
   c. Enter LD_Emp in the Name field.
   d. (Optional) Enter "Employee Group" in the Description field.
   e. Click Apply.
   f. Repeat steps 2b through 2e to create a new group for LD_Mgr.

Figure 30-1 Configuring a New User Group


myrealm> Create Group

Connected to : localhost :7001 | You are logged in as : system | Logout

This page allows you to define a group in this security realm.

Name: LD_Emp

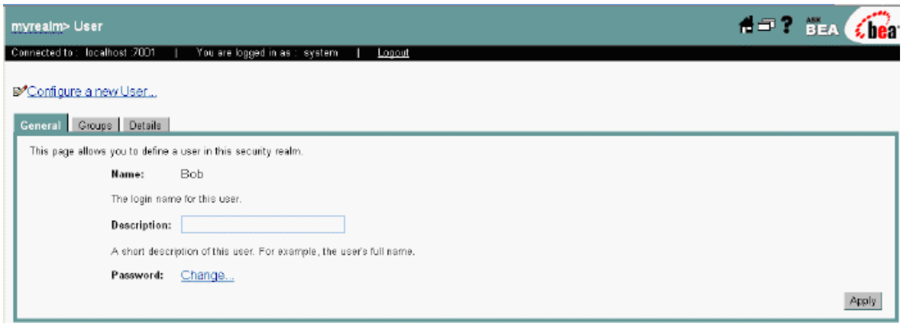
The name for this group

Description: Employee Group

A short description of this group

3. Assign the user Bob to the LD_Emp group, by completing the following steps:
   a. Choose Security → Realms → myrealm → Users.
   b. Click Bob in the User column. The User page for Bob opens.

Figure 30-2 User Page for Bob


myrealm> User
Connected to : localhost :7001 | You are logged in as : system | Logout
Configure a new User...
General
Groups
Details
Apply

This page allows you to define a user in this security realm.

Name: Bob

The login name for this user.

Description:

A short description of this user. For example, the user's full name.

Password: Change...

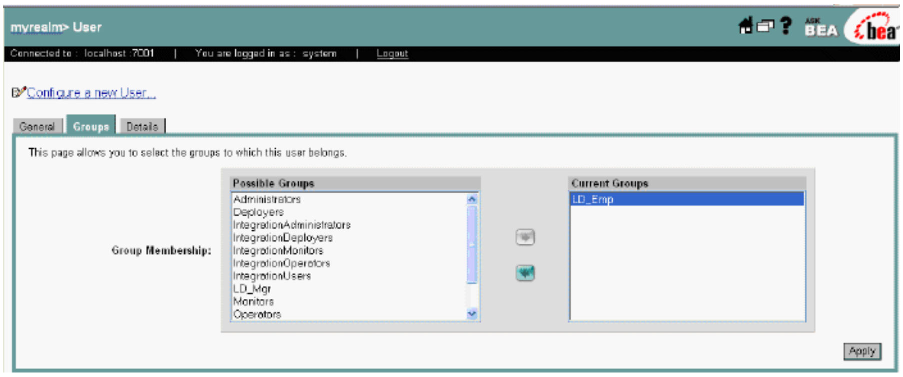
c. Click the Groups tab. The Groups page opens.

d. Select LD_Emp from the Possible Groups pane.

e. Click the arrow (→) to add the group to the Current Groups pane.

f. Click Apply.

Figure 30-3 Group Assignment Page for Bob


myrealm> User
Connected to : localhost :7001 | You are logged in as : system | Logout
Configure a new User...
General
Groups
Details
Apply

This page allows you to select the groups to which this user belongs.

Group Membership:

Possible Groups: Administrators, Deployers, IntegrationAdministrators, IntegrationDeployers, IntegrationMonitors, IntegrationOperators, IntegrationUsers, LD_Mgr, Monitors, Operators

Current Groups: LD_Emp

4. Assign the user Joe in the LD_Mgr group, by completing the following steps:

a. Choose Security → Realms → myrealm → users.

b. Click Joe in the User column. The User page for Joe opens.

c. Click the Groups tab. The Groups page opens.

d. Select LD_Mgr from the Possible Groups pane.

e. Click the arrow (→) to add the group to the Current Groups pane.

f. Click Apply.

30.2 Writing the XQuery Security Function

You can specify a security function using XQuery syntax. In this example, access to social security numbers is restricted to managers.

Objectives

In this exercise, you will:

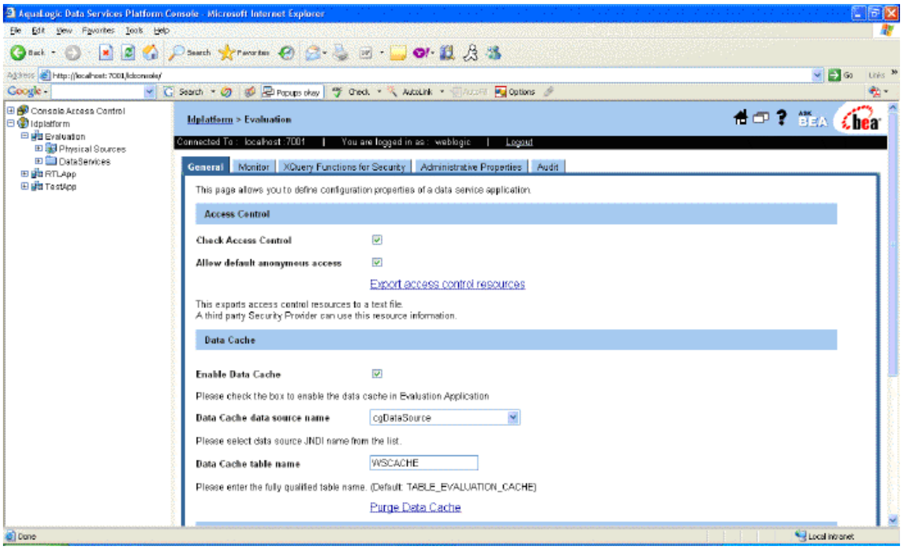
- Set security access control.
- Set a security XQuery function.

Instructions

1. Login to the ALDSP Console (`http://localhost:7001/ldconsole/`), using the following credentials:
   – User Name = weblogic
   – Password = weblogic
2. Using the plus (+) icon, expand the ldplatform directory.

   Note: If you click the ldplatform name, the Application List page opens. You do not want this page for this lesson.
3. Click Evaluation. The Administration Control's General page opens.
4. Select Check Access Control.
5. Select Allow Default Anonymous Access.

Figure 30-4 Setting Access Control



Connected To : localhost:7001 | You are logged in as : weblogic | Logout

This page allows you to define configuration properties of a data service application.

Access Control

Check Access Control ✓

Allow default anonymous access ✓

Export access control resources

This exports access control resources to a text file.
A third party Security Provider can use this resource information.

Data Cache

Enable Data Cache ✓

Please check the box to enable the data cache in Evaluation Application

Data Cache data source name: cgDataSource

Please select data source JNDI name from the list.

Data Cache table name: WSCACHE

Please enter the fully qualified table name. (Default: TABLE_EVALUATION_CACHE)

Purge Data Cache

6. Select Xquery Functions for Security and enter the following function:

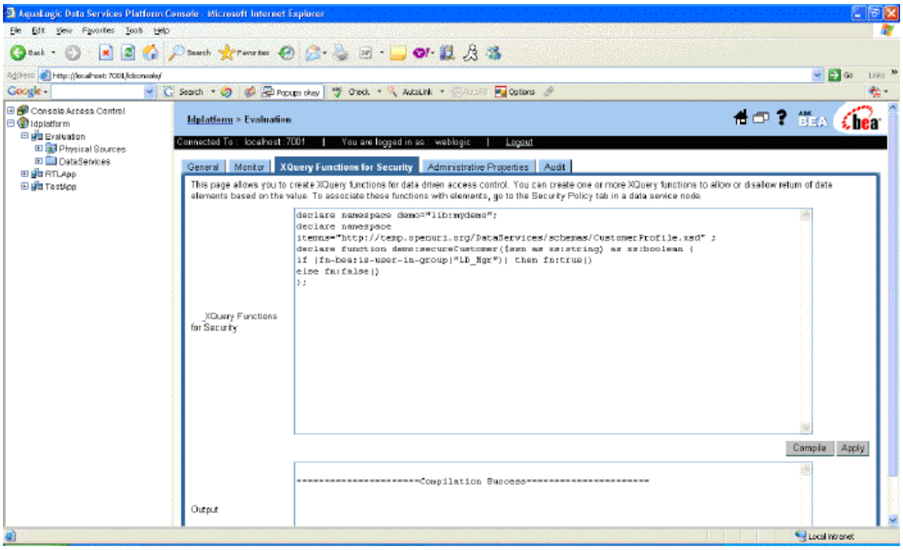
   Note: Namespaces may be different for your application.

```
declare namespace demo="lib:mydemo";

declare namespace
items="http://temp.openuri.org/DataServices/schemas/CustomerProfile.xsd
" ;

declare function demo:secureCustomer($ssn as xs:string) as xs:boolean {

if (fn-bea:is-user-in-group("LD_Mgr")) then fn:true()

else fn:false()

};
```

7. Click Apply.

8. Click Apply again. You should now have the following:

Figure 30-5 Specifying Security XQuery Function Code


ldplatform > Evaluation
Connected To : localhost :7001 | You are logged in as : weblogic | Logout
General
Monitor
XQuery Functions for Security
Administrative Properties
Audit
XQuery Functions for Security
Compile
Apply
Output

This page allows you to create XQuery functions for data driven access control. You can create one or more XQuery functions to allow or disallow return of data elements based on the value. To associate these functions with elements, go to the Security Policy tab in a data service node.

```
declare namespace demo="lib:mydemo";
declare namespace
items="http://temp.openuri.org/DataServices/schemas/CustomerProfile.xsd" ;
declare function demo:secureCustomer($ssn as xs:string) as xs:boolean {
if (fn-bea:is-user-in-group("LD_Mgr")) then fn:true()
else fn:false()
};
```

```
=====================Compilation Success=====================
```

30.3 Activating the XQuery Function for Security

The next step in setting an XQuery security function is to set security at the element level.

Objectives

In this exercise, you will:

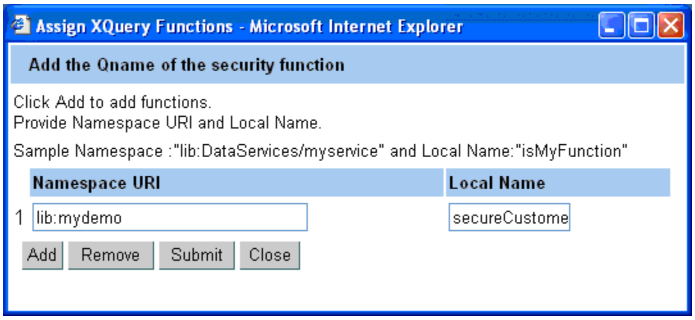
- Secure data source elements.
- Set a security policy.

Instructions

1. In the ALDSP Console expand the Evaluation folder and navigate to the CustomerProfile data service, located in DataServices\CustomerManagement.
2. Navigate to the Security Policy dialog (Admin → Security → Security Policy).
3. Click the icon in the XQuery Function for Security column for the CustomerProfile/customer/ssn resource. The Assign XQuery Functions window opens.

4. Click on Apply. The icon in the Security Policy tab will appear.
5. Set the Namespace URI and Local Name, by completing the following steps:
   a. Click Add and enter the following values:
      - Namespace URI: lib:mydemo
      - Local Name: secureCustomer
   b. Click Submit.
   c. Click Close.

Figure 30-6 QName Information


Assign XQuery Functions - Microsoft Internet Explorer
Add the Qname of the security function

Click Add to add functions.
Provide Namespace URI and Local Name.
Sample Namespace :"lib:DataServices/myservice" and Local Name:"isMyFunction"

	Namespace URI	Local Name
1	lib:mydemo	secureCustome

30.4 Testing the XQuery Security Function

Using the security credentials for Bob and Joe, you can now test the XQuery security function.

Objectives

In this exercise, you will:

- Using two different user logins, test access control.
- View the results.

Instructions

1. Set the login properties to Bob and run a test, by completing the following steps:

a. In the ALDSP-enabled Workshop application, choose Tools → Application Properties → WebLogic Server.

b. Select Use Credentials Below.

c. Enter “Bob” and “password” in the Use Credentials Below fields.

d. Click OK.

e. Open `CustomerProfile.ds` in Test View. (The file is located in the CustomerManagement folder.)

f. Select `getAllCustomers()` from the function drop-down list.

g. Click Execute. All customer data, except SSNs, should be returned.

Note: In order to deploy from WorkShop User/Group you should have permission to deploy applications.

2. Change the login properties to Joe and run a test. All customer data, including SSNs, should be returned.
3. In the ALDSP Console expand the Evaluation folder and navigate to the CustomerProfile data service, located in DataServices\CustomerManagement.
4. Click Security Policy.
5. Click the icon in the XQuery Function for Security column for the CustomerProfile/customer/ssn resource. The QName window opens.
6. Click Remove, click Submit, and then click Close to remove the following:
   - Namespace URI: lib:mydemo;
   - Local Name: secureCustomer

   WARNING: You must remove the Namespace/Local Name information before you can proceed with the following lessons.
7. Click Tools→Application Properties.
8. Use the following credentials:
   - User name = weblogic
   - Password = weblogic

Lesson Summary

In this lesson, you learned how to:

- Establish security based on XQuery functions.
- Write security XQuery functions.

Creating Data Services from Stored Procedures

Enterprise databases utilize stored procedures to improve query performance, manage and schedule data operations, enhance security, and so forth. Stored procedures are essentially database objects that logically group a set of SQL and native database programming language statements together to perform a specific task.

You can import stored procedure metadata from any relational data available to the BEA WebLogic Server. ALDSP then uses that metadata to generate a physical data service that you can then use in logical data services.

Objectives

After completing this lesson, you will be able to:

- Import stored procedures as a Java project within an application.
- Import stored procedure metadata into a data service.

Overview

Imported stored procedure metadata is quite similar to imported metadata for relational tables and views. Stored procedure metadata generally contains:

- A data service file with a pragma that describe the parameters of the stored procedure.
- A schema file with the same primary name as the procedure name.

Note: If a stored procedure includes only one return value and the value is either simple type or a row set that is mapping to an existing schema, no schema file is created.

Handling Stored Procedure Row Sets

A row set type is a complex type, whose name can include:

- The parameter name, if there is an input/output or output only parameter.
- An assigned name such as RETURN_VALUE, if there is a return value.
- The referenced element name (result rowsets) in a user-specified schema.

The row set type contains a repeatable element sequence (for example, called CUSTOMER) with the fields of the row set.

Notes:

- All row set-type definitions must conform to the structure in the stored procedure itself. In some cases the Metadata Import Wizard will be able to automatically detect the structure of a row set and create an element structure. However, if the structure cannot be determined, you will need to provide it through the wizard.
- Each database vendor approaches stored procedures differently. Refer to your database documentation for details on managing stored procedures.
- XQuery support limitations are, in general, due to JDBC driver limitations.
- ALDSP does not support rowset as an input parameter.

31.1 Importing a Stored Procedure into the Application

The first step in demonstrating ALDSP's ability to access data through a stored procedure is to import the procedure into the application.

Objectives

In this exercise, you will:

- Import stored procedures as a Java project.
- Test the results.

Instructions

1. Import storedprocs as a Java project, adding it to the Evaluation application. The project is located in <beahome>\weblogic81\samples\LiquidData\EvalGuide.
2. Build the storedprocs project. The storedprocs.jar file will be added to the Libraries folder.
3. Shutdown the PointBase database, by stopping WebLogic Server.

   Note: Stopping WebLogic Server calls the PointBase shutdown script.
4. Open the `startPointBase.cmd` in a text editor such as Notepad. The file is located in `<beahome>\weblogic81\common\bin`.
5. In the `startPointBase.cmd` script, search for the string "@REM Add PointBase classes to the classpath" and add the complete path of the `storedprocs.jar` file below this line in the script as follows:

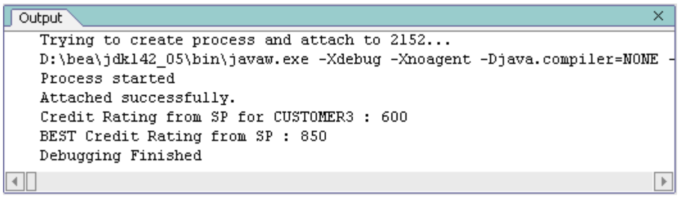
   ```
   set
   CLASSPATH=<beahome>\user_projects\applications\Evaluation\APP-INF\lib\s
   toredprocs.jar;%POINTBASE_CLASSPATH%
   ```

   Note:

   - For reference, the modified `startPointBase.cmd` is included in `samples\liquiddata\EvalGuide`.
   - The CLASSPATH depends on your WebLogic Server installation. User can copy the correct path from the Output window of WebLogic Workshop.
6. Start WebLogic Server, which in turn starts the PointBase database.
7. Run `CreditRatingStoredProcedure.java` to define the stored procedures in PointBase.
8. Click OK at the pop-up message.
9. Confirm that the stored procedure executed, by reviewing the contents in the Output window. You should see the credit rating for CUSTOMER3.

   Note: Your credit rating may be different, based on the changes that you made in Exercise 23.2 Creating an Update Override.

Figure 31-1 Output Window View of Stored Procedures Compilation



```
Trying to create process and attach to 2152...
D:\bea\jdk142_05\bin\javaw.exe -Xdebug -Xnoagent -Djava.compiler=NONE -
Process started
Attached successfully.
Credit Rating from SP for CUSTOMER3 : 600
BEST Credit Rating from SP : 850
Debugging Finished
```

31.2 Importing Stored Procedure Metadata into a Data Service

Importing a stored procedure's source metadata enables the generation of a stored procedure data service.

Objectives

In this exercise, you will:

- Import source metadata into a new data service.
- Test the stored procedure data service.

Instructions

1. Create a new folder in the DataServices project and name it StoredProcedures.
2. Import stored procedures metadata, by completing the following steps:
   a. Right-click the StoredProcedures folder.
   b. Choose Import Source Metadata.
   c. Select Relational from the Data Source Type drop-down list, then click Next.
   d. Select cgDataSource from the Data Source drop-down list, then click Next.
   e. Expand the WEBLOGIC\Procedures folders.
   f. Select GETCREDITRATING_SP, click Add, and click Next.
   g. Accept the default settings displayed in the Configure Procedure window, then click Next.

Note: Do not select GETCREDITRATING_SP as a side-effect procedure.

h. Accept the default settings displayed in the Summary window and click Finish.

3. Build the DataServices project.
4. In the Application pane, confirm that there is a new data service, `GETCREDITRATING_SP.ds`, located in the StoredProcedures folder.
5. Test the data service, by completing the following steps:

    a. Open `GETCREDITRATING_SP.ds` in Test View.

    b. Select `GETCREDITRATING_SP(x1)` from the Function drop-down list.

    c. Enter CUSTOMER3 in the Parameter field.

    d. Click Execute. You should see the credit rating for Customer3

    e. Review the results.

Lesson Summary

In this lesson, you learned how to:

- Import stored procedures into an application.
- Import stored procedure source metadata into a data service.

Creating Data Services from Java Functions

A Java function is another form of metadata that ALDSP can use as a data source. This is perhaps the most powerful metadata, because it allows ALDSP to utilize any data source that can be accessed from Java, such as Enterprise Java Beans, JMS/messaging applications, LDAP and other directory services, text/binary files that can be read through Java I/O, and even DCOM-based applications like Microsoft Excel.

In this lesson, you will access three data sources through Java functions:

- WebLogic's embedded LDAP, by importing a Directory Service Markup Language (DSML)-based Java application as a Java function.
- Data in a Microsoft Excel spreadsheet, by importing a Java application that uses JCOM to access the MS Excel spreadsheet.
- An Enterprise Java Bean that returns customer credit card information using a Java function.

Objectives

After completing this lesson, you will be able to:

- Write Java functions and access them from data services.

Overview

When you use ALDSP's Import Source Metadata feature to import user-defined Java functions, the functions are introspected to create the necessary method signatures and parameter metadata. At the same time, a prologue is created that defines the function's signatures and relevant schema type for complex elements such as Java classes and arrays.

In ALDSP, user-defined functions are treated as Java classes. The following are supported:

- Java primitive types and single-dimension arrays, such as Boolean, byte, and char.
- XMLBean classes corresponding to global elements, complex types, and arrays. The classes generated by XMLBeans can be used as parameters or Return types. The advantage of using XMLBean-generated classes is that you do not need to define a schema for the references complex type or element.

The Metadata Import Wizard supports marshalling and unmarshalling that converts Java token iterators into XML, and vice versa. For example, you start with a Java function, getListGivenMixed, defined as follows:

```
public static float[] getListGivenMixed(float[] fpList, int size) {

int listLen = ((fpList.length > size) ? size : fpList.length);

float fpListop = new float[listLen];

for (int i =0; i < listLen; i++)

fpListop[i]=fpList[i];

return fpListop;

}
```

After the function is processed through the Metadata Import Wizard, the following XML-based metadata is generated:

```
(::pragma xds <x:xds xmlns:x="urn:annotations.ld.bea.com"

targetType="t:float" xmlns:t="http://www.w3.org/2001/XMLSchema">

<javaFunction

classpath="D:\jf\build\jar\jfTest.jar;D:\jf\xbeanTests\xbeangen\

Customer.jar;D:\wls82\weblogic81\server\lib\xbean.jar"

class="jfTest.Customer"/>

</x:xds>::)

declare namespace f1 = "ld:javaFunc/float";

(::pragma function <f:function xmlns:f="urn:annotations.ld.bea.com"
kind="datasource" access="public">
<params>
<param nativeType="[F"/>
<param nativeType="int"/>
</params>
</f:function>::)
declare function f1:getListGivenMixed($x1 as xsd:float*, $x2 as xsd:int)
as xsd:float* external;
```

The corresponding XQuery for the imported Java function would be as follows:

```
declare namespace f1 = "ld:javaFunc/float";
let $y := (2.0, 4.0, 6.0, 8.0, 10.0)
let $x := f1:getListGivenMixed($y, 2)
return $x
```

Note: To ensure successful importation and usage within ALDSP, the Java function should be static functions and its package and class names should be defined in its namespace. ALDSP recognizes the Java method name as the XQuery function name qualified with the Java function namespace.

For detailed information about using Java functions within ALDSP see the Data Services Developer's Guide.

32.1 Accessing Data Using WebLogic's Embedded LDAP Function

ALDSP enables access to data services, using WebLogic's embedded LDAP function. You will learn how to use this functionality by importing a Directory Service Markup Language (DSML)-based Java application as a Java function.

Objectives

In this exercise, you will:

- Set the LDAP security credential for WebLogic's Embedded LDAP.

- Create a new user account.
- Import JAR files and Java applications that will be used to generate a data service.
- Test the data service.

Instructions

1. In the DataServices project, create a folder and name it Functions. This is where you will place the Java functions that you want to import.
2. Set the LDAP security credential for WebLogic's Embedded LDAP, by completing the following steps:
   a. Open the WebLogic Server Console from your browser:

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

   b. Login using the following credentials:
      - User Name = weblogic
      - Password = weblogic

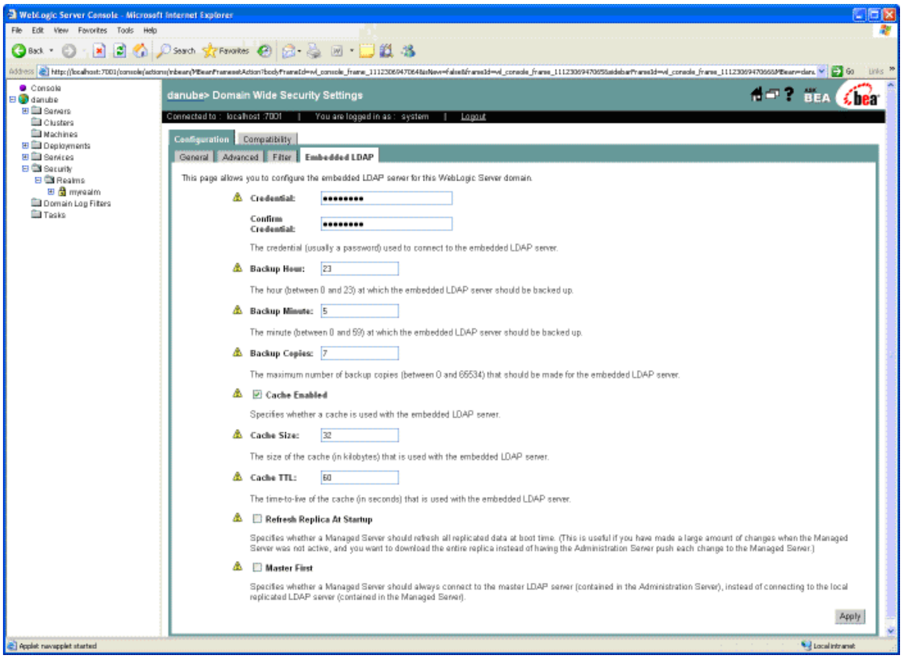
   c. Select the Security folder, located under the ldplatform domain.

   d. Click Embedded LDAP.

   e. Enter security in the Credential and Confirm Credential fields.

   f. Click Apply. This allows access to the WebLogic Server LDAP.

Figure 32-1 Setting LDAP Access Credentials



danube> Domain Wide Security Settings

Connected to : localhost :7001 | You are logged in as : system | Logout

This page allows you to configure the embedded LDAP server for this WebLogic Server domain.

Credential: ●●●●●●●●

Confirm Credential: ●●●●●●●●

The credential (usually a password) used to connect to the embedded LDAP server.

Backup Hour: 23

The hour (between 0 and 23) at which the embedded LDAP server should be backed up.

Backup Minute: 5

The minute (between 0 and 59) at which the embedded LDAP server should be backed up.

Backup Copies: 7

The maximum number of backup copies (between 0 and 65534) that should be made for the embedded LDAP server.

[✓] Cache Enabled

Specifies whether a cache is used with the embedded LDAP server.

Cache Size: 32

The size of the cache (in kilobytes) that is used with the embedded LDAP server.

Cache TTL: 60

The time-to-live of the cache (in seconds) that is used with the embedded LDAP server.

[ ] Refresh Replica At Startup

Specifies whether a Managed Server should refresh all replicated data at boot time. (This is useful if you have made a large amount of changes when the Managed Server was not active, and you want to download the entire replica instead of having the Administration Server push each change to the Managed Server.)

[ ] Master First

Specifies whether a Managed Server should always connect to the master LDAP server (contained in the Administration Server), instead of connecting to the local replicated LDAP server (contained in the Managed Server).

3. You will need to restart the WebLogic Server now as change to this property does not take effect until the Server is restarted.

4. Create a new user, by completing the following steps:

   a. Expand the Security → Realms → myrealm → Users folders.

   b. Click Configure a New User, using your name and a password of your choice.

   c. Click Apply.

5. In WebLogic Workshop right-click the Libraries folder and import all the JAR files located in the samples\liqiddata\EvalGuide\ldap\lib folder into the Libraries folder in Workshop.

6. Right-click the Functions folder and import `DSML.java` from the samples\liquiddata\EvalGuide folder.

7. Build the DataServices project.

8. Import the Java function metadata for the DSML Java application into the Functions folder by completing the following steps:
   a. Right click the Functions folder and choose Import Source Metadata.
   b. Select Java Function for the Data Source Type and click Next.
   c. In the Class Name field, browse and select DataServices.jar\Functions\DSML.class and click Next.
   d. Select the `callDSML()` function, click Add, and then click Next.

   Note: Do not select the callDSML procedure as a side-effect procedure.

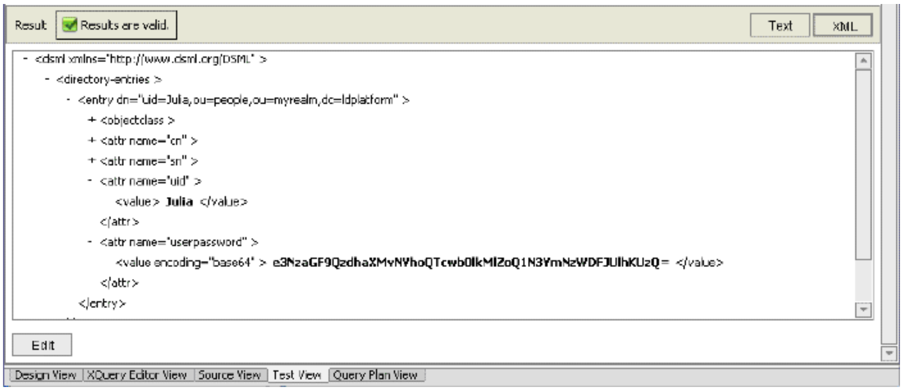
   e. Accept the default settings in the Summary window and click Finish.

      The `dsml.ds` file and schemas folder are added to the Functions folder.
9. Build the DataServices project.
10. Test the DSML data service by completing the following steps:
    a. Open `dsml.ds` in Test View.
    b. Select `callDSML()` from the Function drop-down list.
    c. Enter the following arguments (for more information on LDAP arguments and access, see http://dev2dev.bea.com/codelibrary/code/ld_ldap.jsp):

Description	Argument
LDAP URL	ldap://localhost:7001
Principal (Directory Manager)	cn=Admin
Credentials (Password)	security
JNDI (true: use JNDI to access LDAP; false: use native LDAP connection	jndi
Base domain name to search	dc=ldplatform
Filter used to search	cn=<your user name>

    d. Click Execute.
    e. View the results.

Figure 32-2 Results for callDSML()



```
- <dsml xmlns="http://www.dsml.org/DSML" >
  - <directory-entries >
    - <entry dn="uid=Julia,ou=people,ou=myrealm,dc=ldplatform" >
      + <objectclass >
      + <attr name="cn" >
      + <attr name="sn" >
      - <attr name="uid" >
          <value> Julia </value>
        </attr>
      - <attr name="userpassword" >
          <value encoding="base64" > e3NzaGF9QzdhaXMvNVhoQTcwb0lkMlZoQ1N3YmNzWDFJUlhKUzQ= </value>
        </attr>
      </entry>
```

32.2 Accessing Excel Spreadsheet Data Using JCOM

Data in a Microsoft Excel spreadsheet can be accessed through JCOM.

Objectives

In this exercise, you will:

- Import JAR and Java files appropriate that will be used to generate a data service for using JCOM.
- Test the results.

Instructions

1. Right-click the Libraries folder and using the add Add to Library option, add all the JAR files located in the samples\liqiddata\EvalGuide\excel\lib folder.
2. Right-click the Functions folder and import `excel_jcom.java` from <beahome>\weblogic81\samples\LiquidData\EvalGuide.
3. Build the DataServices project.
4. Import the Java function metadata for the Excel JCOM Java application into the Functions folder, by completing the following steps:
   a. Right-click the Functions project and choose Import Source Metadata.

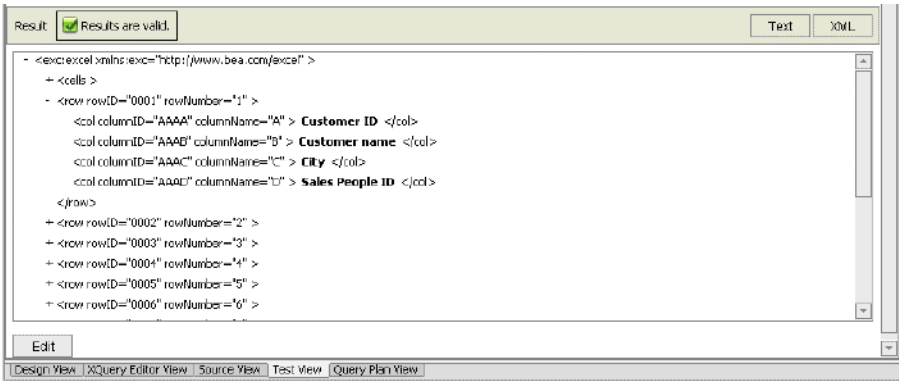
   b. Select Java Function for the Data Source Type and click Next.
   c. In the Class Name field, browse and select DataServices\Functions\Functions.excel_jcom and then click Next.
   d. Select the `getExcel()` function, click Add, and then click Next.
   e. Accept the default setting in the Select Side Effect Procedures window and click Next.
   f. Accept the default settings in the Summary window and click Finish. The excel.ds and associated schema files are added to the Functions folder.

5. Build the DataServices project.
6. Test the Excel data service, by completing the following steps:
   a. Open `excel.ds` in Test View.
   b. Select `getExcel(x1, x2)` from the Function drop-down list.
   c. Enter the following arguments:

Description	Argument
XLS File Name	`<beahome>\weblogic81\samples\LiquidData\EvalGuide\excel\test.xls`
Worksheet Name	Customers

7. Review the results.

Figure 32-3 Results of the getExcel function



```
- <exc:excel xmlns:exc="http://www.bea.com/excel" >
  + <cells >
  - <row rowID="0001" rowNumber="1" >
      <col columnID="AAAA" columnName="A" > Customer ID </col>
      <col columnID="AAAB" columnName="B" > Customer name </col>
      <col columnID="AAAC" columnName="C" > City </col>
      <col columnID="AAAD" columnName="D" > Sales People ID </col>
    </row>
  + <row rowID="0002" rowNumber="2" >
  + <row rowID="0003" rowNumber="3" >
  + <row rowID="0004" rowNumber="4" >
  + <row rowID="0005" rowNumber="5" >
  + <row rowID="0006" rowNumber="6" >
```

Note: For more information on Excel access refer to a dev2dev sample illustrating accessing data in an MS-Excel spreadsheet. As of this writing the sample is located at:

`http://codesamples.projects.dev2dev.bea.com/servlets/Scarab?id=S230`

32.3 (Optional) Accessing Data Using an Enterprise Java Bean

Create an Enterprise Java Bean that returns customer credit card information using a Java function.

Objectives

In this exercise, you will:

- Import the schemas needed to define an EJB-based data service.
- Generate an EJB-based data service.
- Test the EJB-based data service.

Instructions

1. Create a Schemas Project, by completing the following steps:

   a. Right-click the Evaluation application folder and import the Schemas folder as a Schema Project. The folder is located in:

`<beahome>\weblogic81\samples\LiquidData\EvalGuide\ejb`

This schema will be used for the EJB results, which returns an XML document containing credit card information for a customer.

b. Build the Schemas project.

2. Create an EJB Project, by completing the following steps:

   a. Right-click the Evaluation application folder and import the EJB folder as an EJB Project. The folder is located in:

      `<beahome>\weblogic81\samples\LiquidData\EvalGuide\ejb`. This contains:

      - A container-managed entity bean that maps to the credit card database table.
      - A stateless session bean that invokes the entity bean finder method returning a list of credit cards for a given customer in the shape of the CREDIT_CARDS XML schema.

   b. Build the EJB project.

3. Create a Java project, by completing the following steps:

   a. Right-click the Evaluation application folder and import the EJBClient folder as a Java Project. The folder is located in:

      `<beahome>\weblogic81\samples\LiquidData\EvalGuide\ejb`

      This project contains the Java client that connects remotely to the stateless session bean. This will be used as the custom function.

   b. Build the EJBClient project.

4. Run `CreditCardClient.java`, which is located in the EJBClient project folder. A list of credit cards for CUSTOMER3 should display in the Output window

   Note: Click OK for the pop-up message. Drag and drop the `CreditCardClient.java` into the Functions folder.

5. Build the DataServices project.

6. Import the Java function metadata for the EJB Client into the DataServices project by completing the following steps.

   a. Right-click the Functions folder and select Import Source Metadata.

   b. Select Java Function as the Data Source Type and click Next.

c. Browse and select `DataServices\Functions.CreditCardClient` as the Class Name and click Next.

d. Select getCreditCards, click Add, and then click Next.

e. Accept the default settings in the Select Side Effect Procedures window.

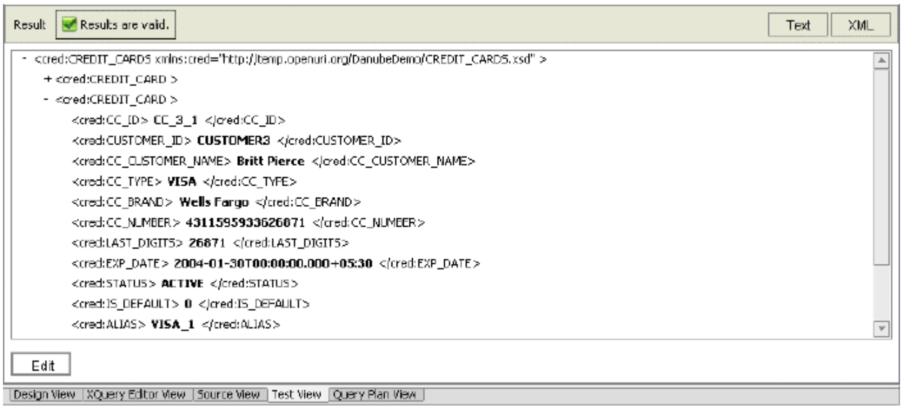
f. Accept the default settings in the Summary window and click Finish. The CREDIT_CARDS.ds file is added to the Functions folder.

Note: Do not confuse this data service with the `CREDIT_CARD.ds` created from the relationship database.

g. Build the DataServices project.

7. Test the `getCreditCards()` function within the CREDIT_CARDS data service. Use CUSTOMER3 as the argument. Confirm that you can retrieve credit card information for Britt Pierce.

Figure 32-4 Results for the getCreditCards() function



```
- <cred:CREDIT_CARDS xmlns:cred="http://temp.openuri.org/DanubeDemo/CREDIT_CARDS.xsd" >
   + <cred:CREDIT_CARD >
   - <cred:CREDIT_CARD >
        <cred:CC_ID> CC_3_1 </cred:CC_ID>
        <cred:CUSTOMER_ID> CUSTOMER3 </cred:CUSTOMER_ID>
        <cred:CC_CUSTOMER_NAME> Britt Pierce </cred:CC_CUSTOMER_NAME>
        <cred:CC_TYPE> VISA </cred:CC_TYPE>
        <cred:CC_BRAND> Wells Fargo </cred:CC_BRAND>
        <cred:CC_NUMBER> 4311595933626871 </cred:CC_NUMBER>
        <cred:LAST_DIGITS> 26871 </cred:LAST_DIGITS>
        <cred:EXP_DATE> 2004-01-30T00:00:00.000+05:30 </cred:EXP_DATE>
        <cred:STATUS> ACTIVE </cred:STATUS>
        <cred:IS_DEFAULT> 0 </cred:IS_DEFAULT>
        <cred:ALIAS> VISA_1 </cred:ALIAS>
```

Lesson Summary

In this lesson, you learned how to import the following sources as Java functions:

- WebLogic's embedded LDAP through a Directory Service Markup Language (DSML)-based Java application
- Data in a Microsoft Excel spreadsheet through a Java application that uses JCOM to access the MS Excel spreadsheet.

- An Enterprise Java Bean that returns customer credit card information.

Creating Data Services from XML Files

XML documents are a convenient means for handling hierarchical data. ALDSP enables the creation of data services that read data stored in XML files.

Objectives

After completing this lesson, you will be able to:

- Import XML metadata and query XML files.
- Confirm that the results conform to the XML file specifications.

Overview

Contents of an XML file can be turned into a data service and used as a data source.

In this lab you will create a data service that queries data stored in an XML file. The XML file contains UNSPSC product category received from third-party vendor.

33.1 Importing XML Metadata and XML Schema Definition

Importing XML metadata and schema definitions is similar to importing relational and Web service metadata, with some differences.

Objectives

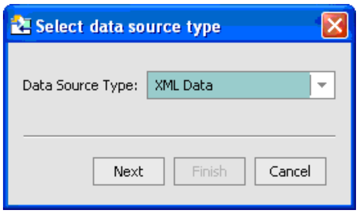
In this lab, you will:

- Import XML metadata.
- Associate a schema and XML source file with the data service.
- Generate a data service that reads XML data for the UNSPSC product category.

Instructions

1. Import the XMLFiles folder into the DataServices project. The folder is located in <beahome>\weblogic81\samples\LiquidData\EvalGuide.
2. Right click the XMLFiles folder and select Import Source Metadata.
3. Select XML Data from the Data Source Type drop-down list, then click Next.

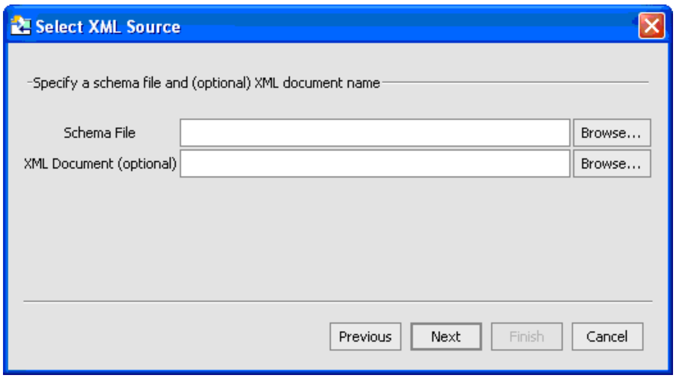
Figure 33-1 Import XML Data



Data Source Type: XML Data

The Select XML Source window opens.

Figure 33-2 Select XML Source Window



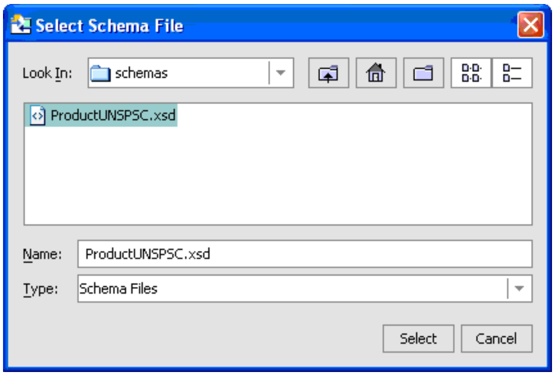
4. Associate a schema file with the data service, by completing the following steps:

a. Click Browse, next to the Schema File field. The XMLFiles directory opens in the Select Schema Files window.

b. Expand the Schemas folder.

c. Select `ProductUNSPSC.xsd` and click Select.

Figure 33-3 Select Schema File

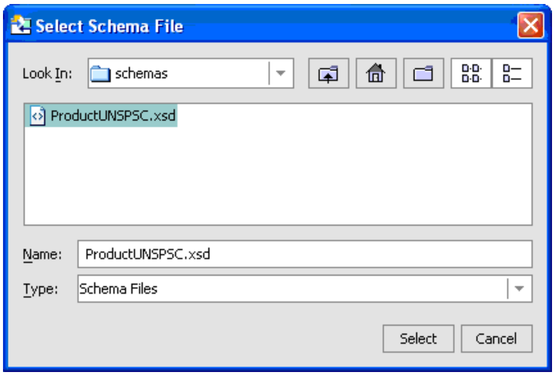


5. Associate the XML Document with the data service, by completing the following steps:

   a. Click Browse, next to the XML Document field. The XMLFiles directory opens in the Select XML Source File window.

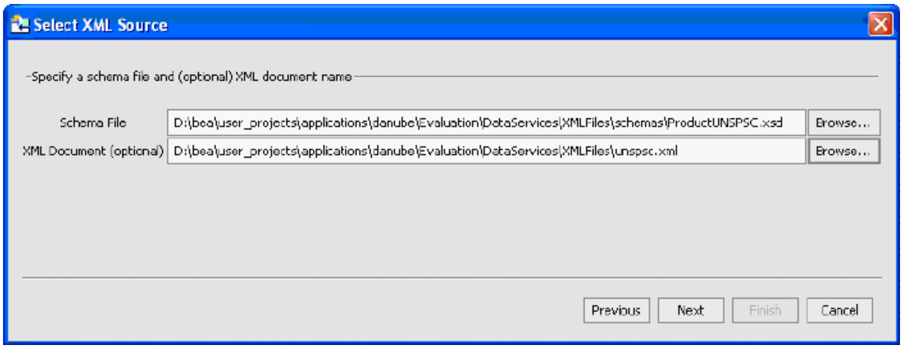
   b. Select unspsc.xml and click Select.

Figure 33-4 Select XML Source File



The Select XML Source window is now populated with file information.

Figure 33-5 Populated Select XML Source Window

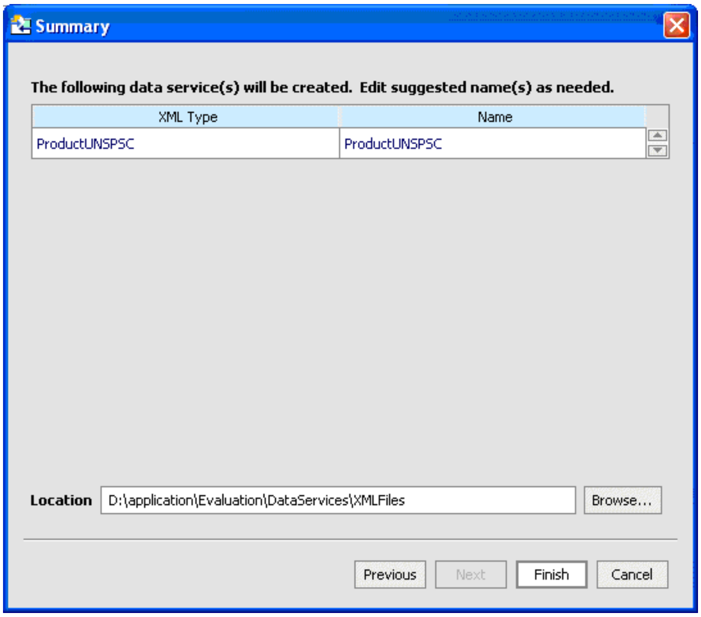


Schema File: D:\bea\user_projects\applications\danube\Evaluation\DataServices\XMLFiles\schemas\ProductUNSPSC.xsd

XML Document (optional): D:\bea\user_projects\applications\danube\Evaluation\DataServices\XMLFiles\unspsc.xml

6. Click Next. The Summary window opens.

Figure 33-6 Summary Window



The following data service(s) will be created. Edit suggested name(s) as needed.

XML Type	Name
ProductUNSPSC	ProductUNSPSC

Location: D:\application\Evaluation\DataServices\XMLFiles

The Summary information includes the following details:

  - XML Type, for XML objects whose source metadata will be imported.
  - Name, for each data service that will be generated from the source metadata. (Any name conflicts appear in red; you can modify any data service name to correct an error condition or to change to a different project-unique name.)
  - Location, where the generated data service(s) will reside.

7. Click Finish. A new data service, called `ProductUNSPSC.ds`, is created in:

   `DataServices\XMLFiles`

33.2 Testing the XML Data Service

After creating an XML data service, you need to confirm that the service is able to return data, based on the associated XML source file.

Objectives

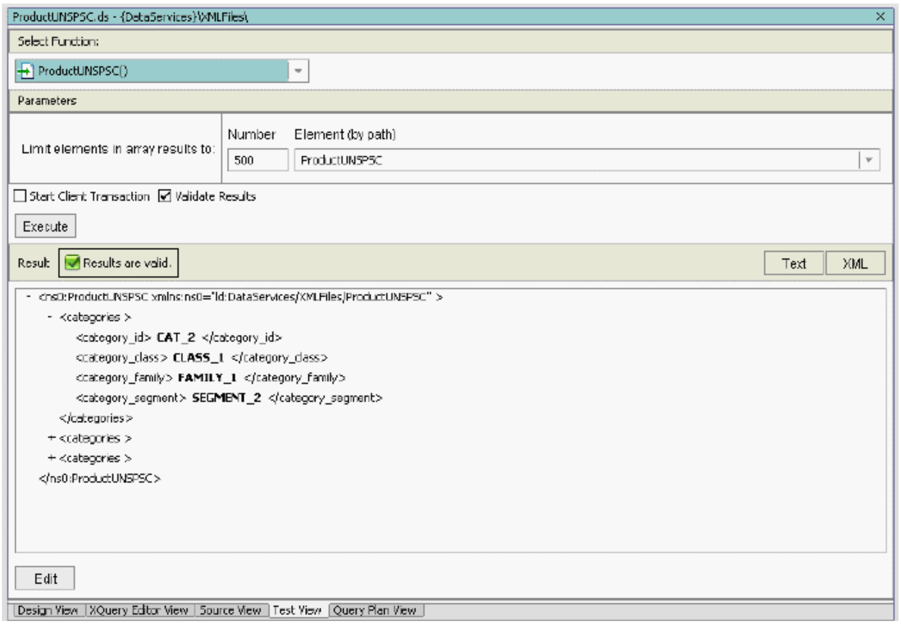
In this lab, you will:

- Build the DataService project.
- Execute the `productUNSPSC()` function.
- Compare the test results with the unspsc.xml file.

Instructions

1. Build the project containing the ProductUNSPSC data service.
2. Open `ProductUNSPSC.ds` in Test View.
3. Test the data service by completing the following steps:
   a. Select `productUNSPSC()` from the Function drop-down list.
   b. Click Execute.
   c. Confirm that you can retrieve data, as displayed in Figure 33-7.

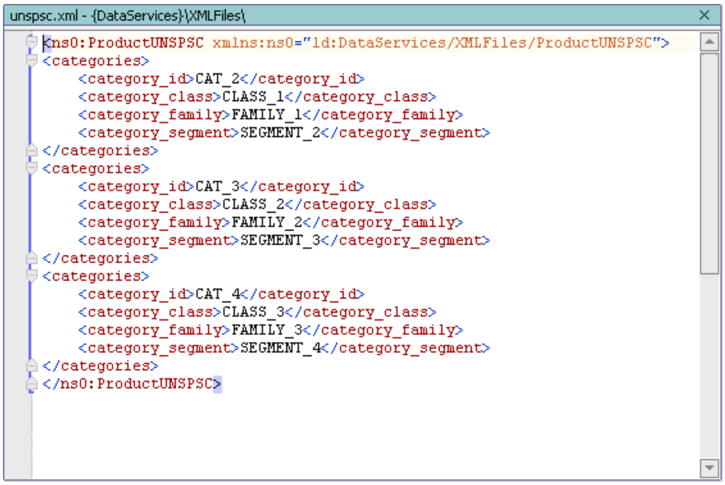
Figure 33-7 XML Data Service Test Results



```
- <ns0:ProductUNSPSC xmlns:ns0="ld:DataServices/XMLFiles/ProductUNSPSC" >
   - <categories >
        <category_id> CAT_2 </category_id>
        <category_class> CLASS_1 </category_class>
        <category_family> FAMILY_1 </category_family>
        <category_segment> SEGMENT_2 </category_segment>
     </categories>
   + <categories >
   + <categories >
  </ns0:ProductUNSPSC>
```

4. 4.In the Application pane expand the XMLFiles folder and open the unspsc.xml file.
5. Confirm that the test results conform to the specifications in the XML file.

Figure 33-8 XML Elements



```
<ns0:ProductUNSPSC xmlns:ns0="ld:DataServices/XMLFiles/ProductUNSPSC">
<categories>
    <category_id>CAT_2</category_id>
    <category_class>CLASS_1</category_class>
    <category_family>FAMILY_1</category_family>
    <category_segment>SEGMENT_2</category_segment>
</categories>
<categories>
    <category_id>CAT_3</category_id>
    <category_class>CLASS_2</category_class>
    <category_family>FAMILY_2</category_family>
    <category_segment>SEGMENT_3</category_segment>
</categories>
<categories>
    <category_id>CAT_4</category_id>
    <category_class>CLASS_3</category_class>
    <category_family>FAMILY_3</category_family>
    <category_segment>SEGMENT_4</category_segment>
</categories>
</ns0:ProductUNSPSC>
```

Lesson Summary

In this lesson, you learned how to:

- Access data in an XML file.
- Confirm that the results conform to the contents of the XML file.

Creating Data Services from Flat Files

Flat files, such as spreadsheets, offer a highly adaptable means of storing and manipulating data, especially data that needs to be quickly changed. Flat files are simply treated as another data source that ALDSP can use to generate metadata and create a data service.

Objectives

After completing this lesson, you will be able to:

- Create a data service that can access data stored in a flat file.
- Associate the flat file data service with a logical data service.

Overview

Flat files, such as spreadsheets, often support a text format called CSV or Comma Separated Values. Such file formats typically have a .csv extension.

34.1 Importing Flat File Metadata

The flat file must be in a ALDSP project, before a data service can be generated. As part of the import process, you must provide a schema name, a file name, or both.

Objectives

In this exercise, you will:

- Create a data service that queries data stored in a flat file. The flat file contains customer valuation data received from an internal department that deals with customer scoring and valuation models. The file contains the following fields:
- Customer_id
- Valuation_date
- Valuation_score

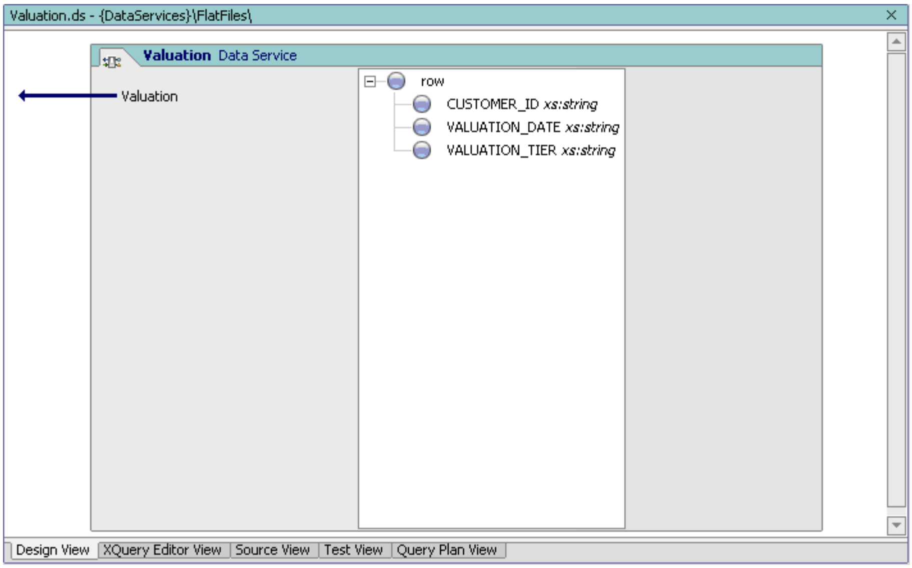
Instructions

1. Right-click the DataServices folder and import the FlatFiles folder, which is located in <beahome>\weblogic81\samples\LiquidData\EvalGuide.
2. Import source metadata by completing the following steps:
   a. Right-click the FlatFiles folder and select Import Source Metadata.
   b. Select Delimited Data from the Data Source Type drop-down list, then click Next.
   c. Ignore the Schema field.
   d. Click Browse, next to the Delimited Source field.
   e. Select Valuation.csv and click Select.
   f. Confirm that the Has Header checkbox is enabled.

      By selecting this option, you specify that the header data, which is usually located in the first row of the spreadsheet, will not be treated as data within the generated data service.
   g. Confirm that the Delimited radio button is enabled. By enabling this option, you specify that the data is separated by a specific character, rather than a fixed width such as 10 spaces.
   h. Confirm that a comma (,) is in the Delimiter field. If data is delimited, then you must specify what character is used to delimit the data. Although the default is a comma, any ASCII character is supported.
   i. Click Next. The Summary dialog box opens.
   j. Click Finish. A new data service called `Valuation.ds` is created in the DataServices\FlatFiles.
3. Open the `Valuation.ds` file in Design View.

4. Open `Valuation.ds` in Design View and confirm that there is a Valuation function. This function will retrieve all data from the flat file.

Figure 34-1 Design View of the Data Service Based on a Flat File



34.2 Testing Your Flat File Data Service

After creating the data service, you need to confirm that the service is able to return data, based on the associated delimited source file.

Objectives

In this exercise, you will:

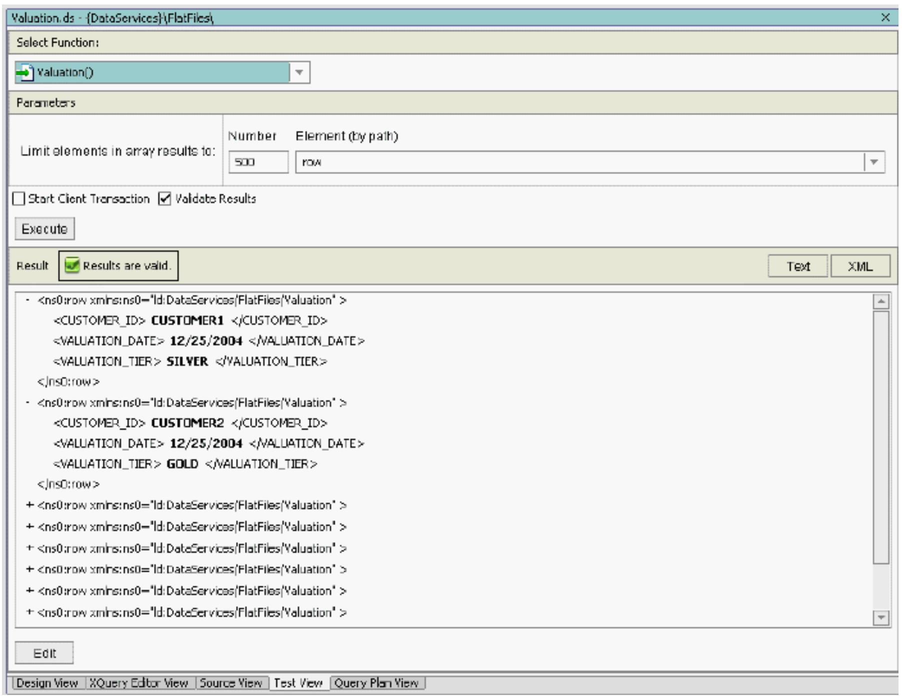
- Build the DataService project.
- Execute the Valuation function.

Instructions

1. Right-click the DataServices folder.

2. Choose Build DataServices.
3. Open `Valuation.ds` in Test View.
4. Test the data service by completing the following steps:
   a. Select `Valuation()` from the Function drop-down list.
   b. Click Execute.
5. Confirm that you can retrieve data, as displayed in Figure 34-2. Notice that the return element is introspected. That is based on the header information in the `Valuation.csv` file.

Figure 34-2 Test Results—Flat File Data Service



```
- <ns0:row xmlns:ns0="ld:DataServices/FlatFiles/Valuation" >
     <CUSTOMER_ID> CUSTOMER1 </CUSTOMER_ID>
     <VALUATION_DATE> 12/25/2004 </VALUATION_DATE>
     <VALUATION_TIER> SILVER </VALUATION_TIER>
  </ns0:row>
- <ns0:row xmlns:ns0="ld:DataServices/FlatFiles/Valuation" >
     <CUSTOMER_ID> CUSTOMER2 </CUSTOMER_ID>
     <VALUATION_DATE> 12/25/2004 </VALUATION_DATE>
     <VALUATION_TIER> GOLD </VALUATION_TIER>
  </ns0:row>
+ <ns0:row xmlns:ns0="ld:DataServices/FlatFiles/Valuation" >
+ <ns0:row xmlns:ns0="ld:DataServices/FlatFiles/Valuation" >
+ <ns0:row xmlns:ns0="ld:DataServices/FlatFiles/Valuation" >
+ <ns0:row xmlns:ns0="ld:DataServices/FlatFiles/Valuation" >
+ <ns0:row xmlns:ns0="ld:DataServices/FlatFiles/Valuation" >
+ <ns0:row xmlns:ns0="ld:DataServices/FlatFiles/Valuation" >
```

34.3 Integrating Flat File Valuation with a Logical Data Service

At this point, you are able to pull data from the flat file. However, integrating the flat file data service into a logical data service lets you retrieve multiple sources of information.

Objectives

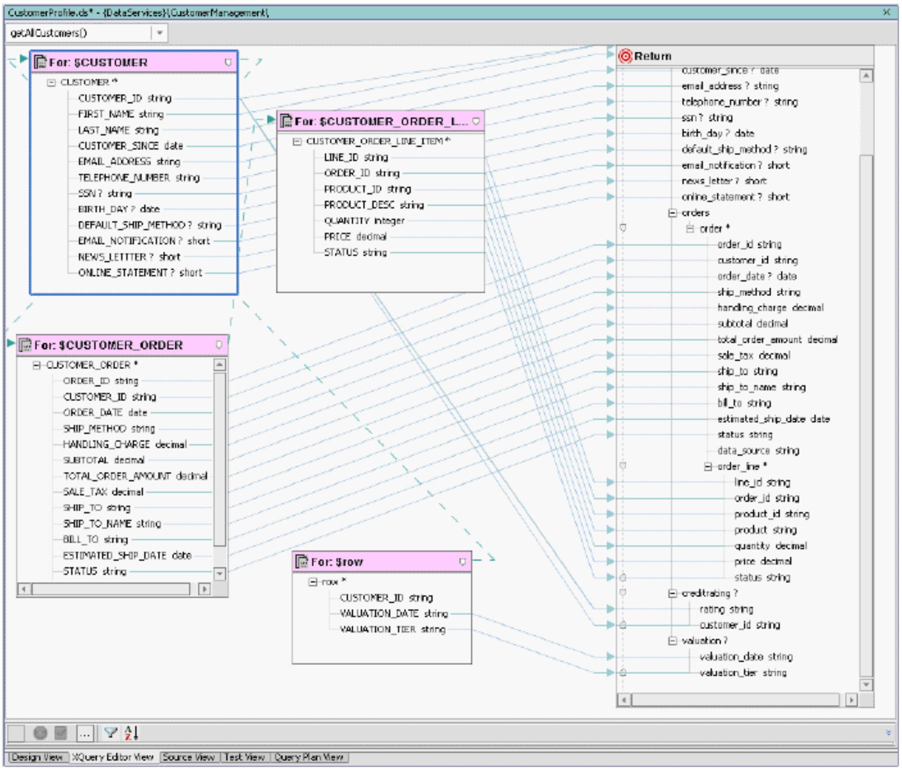
In this exercise, you will:

- Modify a function to retrieve data from a flat file physical data service.
- View the results in both XQuery Editor View and Source View.

Instructions

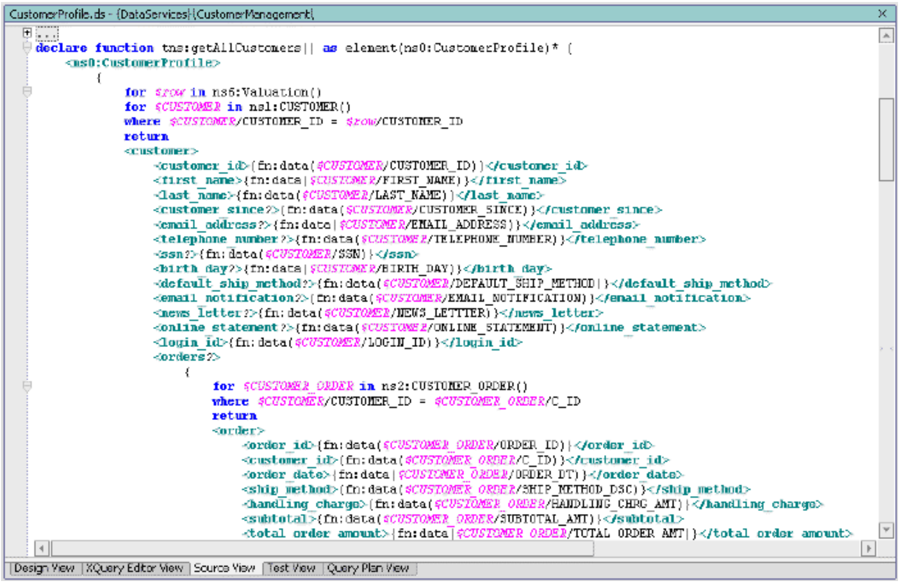
1. Open `CustomerProfile.ds` under DataServices/CustomerManagement/CustomerProfile in XQuery Editor View.
2. Select `getAllCustomers()` from the Function drop-down list.
3. In the Data Services Palette, expand the FlatFiles and `Valuation.ds` folders.
4. Drag and drop `Valuation()` into XQuery Editor View.
5. Create a simple mapping by dragging and dropping the VALUATION_DATE and VALUATION_TIER elements (valuation node) onto the corresponding elements in the Return type.
6. Create a join. Drag and drop the CUSTOMER_ID element (Customer node) onto the corresponding element in the Valuation node. The final layout should be similar to that shown in Figure 34-3:

Figure 34-3 XQuery Editor View of Flat File Data Service Integrated with Logical Data Service



7. Open `CustomerProfile.ds` in Source View and confirm that the following mapping have been created:

Figure 34-4 Source View of Flat File Data Service Integrated with Logical Data Service



```
...
declare function tns:getAllCustomers() as element(ns0:CustomerProfile)* {
    <ns0:CustomerProfile>
        {
            for $row in ns6:Valuation()
            for $CUSTOMER in ns1:CUSTOMER()
            where $CUSTOMER/CUSTOMER_ID = $row/CUSTOMER_ID
            return
            <customer>
                <customer_id>{fn:data($CUSTOMER/CUSTOMER_ID)}</customer_id>
                <first_name>{fn:data($CUSTOMER/FIRST_NAME)}</first_name>
                <last_name>{fn:data($CUSTOMER/LAST_NAME)}</last_name>
                <customer_since?>{fn:data($CUSTOMER/CUSTOMER_SINCE)}</customer_since>
                <email_address?>{fn:data($CUSTOMER/EMAIL_ADDRESS)}</email_address>
                <telephone_number?>{fn:data($CUSTOMER/TELEPHONE_NUMBER)}</telephone_number>
                <ssn?>{fn:data($CUSTOMER/SSN)}</ssn>
                <birth_day?>{fn:data($CUSTOMER/BIRTH_DAY)}</birth_day>
                <default_ship_method?>{fn:data($CUSTOMER/DEFAULT_SHIP_METHOD)}</default_ship_method>
                <email_notification?>{fn:data($CUSTOMER/EMAIL_NOTIFICATION)}</email_notification>
                <news_letter?>{fn:data($CUSTOMER/NEWS_LETTTER)}</news_letter>
                <online_statement?>{fn:data($CUSTOMER/ONLINE_STATEMENT)}</online_statement>
                <login_id>{fn:data($CUSTOMER/LOGIN_ID)}</login_id>
                <orders?>
                    {
                        for $CUSTOMER_ORDER in ns2:CUSTOMER_ORDER()
                        where $CUSTOMER/CUSTOMER_ID = $CUSTOMER_ORDER/C_ID
                        return
                        <order>
                            <order_id>{fn:data($CUSTOMER_ORDER/ORDER_ID)}</order_id>
                            <customer_id>{fn:data($CUSTOMER_ORDER/C_ID)}</customer_id>
                            <order_date>{fn:data($CUSTOMER_ORDER/ORDER_DT)}</order_date>
                            <ship_method>{fn:data($CUSTOMER_ORDER/SHIP_METHOD_DSC)}</ship_method>
                            <handling_charge>{fn:data($CUSTOMER_ORDER/HANDLING_CHRG_AMT)}</handling_charge>
                            <subtotal>{fn:data($CUSTOMER_ORDER/SUBTOTAL_AMT)}</subtotal>
                            <total_order_amount>{fn:data($CUSTOMER_ORDER/TOTAL_ORDER_AMT)}</total_order_amount>
```

34.4 Testing an Integrated Flat File Data Service

Testing the function lets you confirm that the data is correctly retrieved.

Objectives

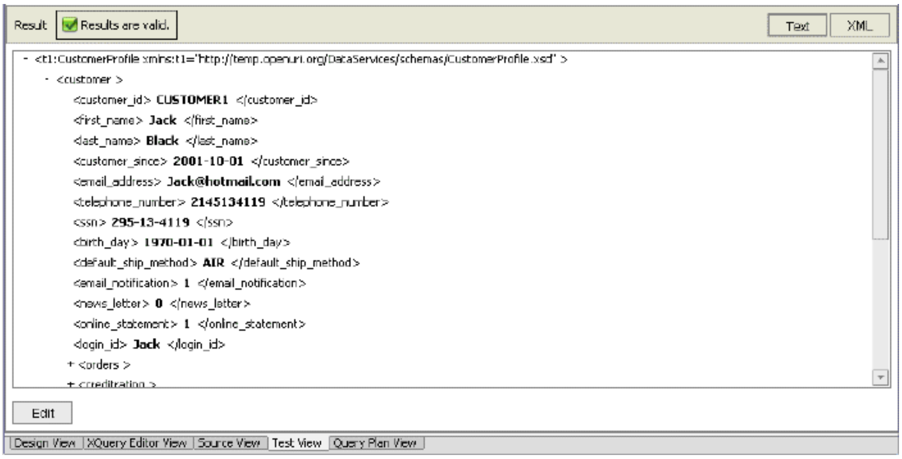
In this exercise, you will:

- Test the getAllCustomers function.
- View the results.

Instructions

1. Open `CustomerProfile.ds` in Test View.
2. Select `getAllCustomers()` from the Function drop-down list.
3. Click Execute.
4. Confirm that you can retrieve valuation information.

Figure 34-5 Test View of Integrated Flat File Data Service



```
- <t1:CustomerProfile xmlns:t1="http://temp.openuri.org/DataServices/schemas/CustomerProfile.xsd" >
  - <customer >
      <customer_id> CUSTOMER1 </customer_id>
      <first_name> Jack </first_name>
      <last_name> Black </last_name>
      <customer_since> 2001-10-01 </customer_since>
      <email_address> Jack@hotmail.com </email_address>
      <telephone_number> 2145134119 </telephone_number>
      <ssn> 295-13-4119 </ssn>
      <birth_day> 1970-01-01 </birth_day>
      <default_ship_method> AIR </default_ship_method>
      <email_notification> 1 </email_notification>
      <news_letter> 0 </news_letter>
      <online_statement> 1 </online_statement>
      <login_id> Jack </login_id>
    + <orders >
    + <creditrating >
```

5. (Optional) Use the getCustomerProfile function, enter CUSTOMER3 in the Parameter field, and click Execute.

   Note: Ensure that the user has access to run the getCustomerProfile function by checking the security settings in the ALDSP Console.

Lesson Summary

In this lesson, you learned how to:

- Import a CSV file containing valuation information.
- Create a flat file physical data service.
- Integrate the flat file physical data service with a logical data service.

Creating an XQuery Function Library

In any ALDSP project you can create XQuery libraries containing functions which can be used by any data service in your application. An XQuery function library is ideal for containing transformation and other types of functions without the overhead of having to build a data service. An XQuery function library can also be used to hold security functions which, in turn, can be used by any data service.

Objectives

After completing this lesson, you will be able to:

- Create and use XFL functions.
- View the results.

Overview

An XQuery Function Library (XFL) contains user functions that return discrete values, such as string, integer, or calendar. These functions are useful for data manipulation at query execution time.

35.1 Creating an XQuery Function Library

In this lesson, you will "encrypt" a customer's SSN to hide its value. As part of this process you will be modifying the `getCustomerProfile()` query function.

Objectives

In this exercise, you will:

- Import a Java file into the DataServices project.
- Import source metadata.
- Test the function

Instructions

1. Create a new folder in the DataServices project and name it xfl.
2. Import `protectSSN.java` in the xfl folder. The file is located in:

   `samples\liquiddata\EvalGuide`

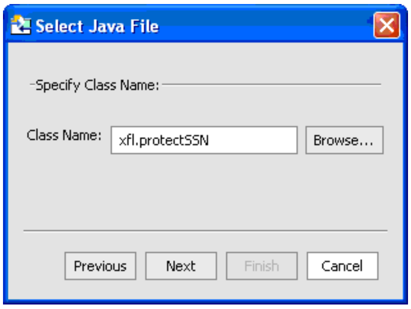
3. Build the DataServices project.
4. Import source metadata into the xfl folder by completing the following steps:

   a. Right-click the xfl folder and choose Import Source Metadata.

   b. Select Java Function from the Data Source Type drop-down list and click Next.

   c. Browse and select DataServices\xfl.protectSSN in the Class Name field and click Next.

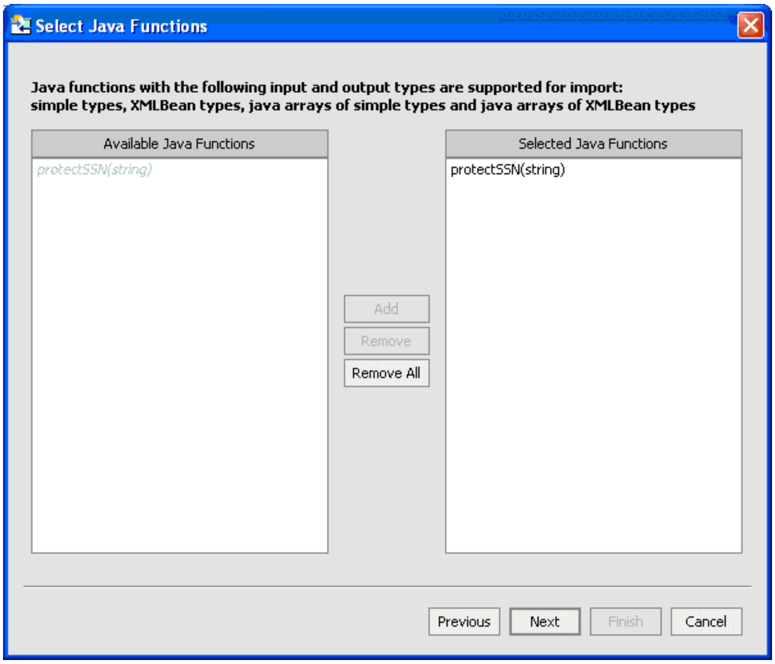
Figure 35-1 Selecting the Java File



Class Name: xfl.protectSSN

   d. Select the protectSSN function, and then click Add.

Figure 35-2 Selecting the Java Function



Java functions with the following input and output types are supported for import:
simple types, XMLBean types, java arrays of simple types and java arrays of XMLBean types

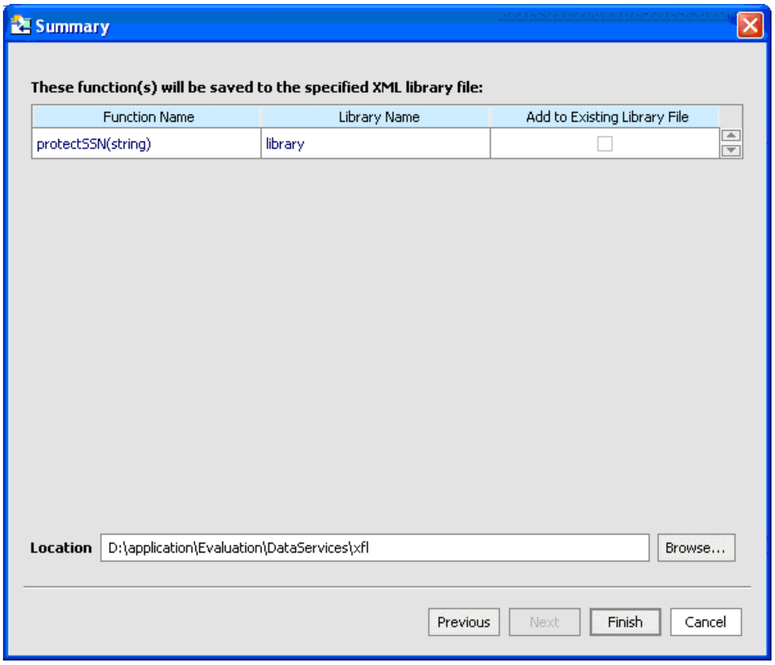
protectSSN(string)

protectSSN(string)

e. Accept the default settings in the Select Side Effect Procedures window and click Next.

f. Click Next. The Summary window opens.

Figure 35-3 imported Java Metadata Summary



These function(s) will be saved to the specified XML library file:

Function Name	Library Name	Add to Existing Library File
protectSSN(string)	library	☐

Location D:\application\Evaluation\DataServices\xfl

g. Click Finish.

5. Test the function, by completing the following steps:

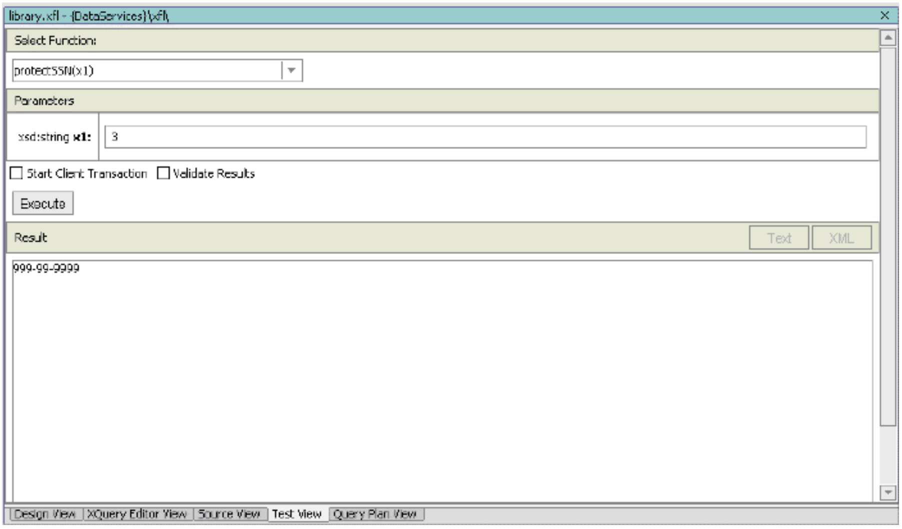
   a. Open `library.xfl` in Test View.

   b. Select protectSSN from the Function drop-down list.

   c. Insert any number in the Parameter field; for example, 3.

   d. Click Execute. The test should return 999-99-9999, regardless of the input parameter.

Figure 35-4 XQuery Function Library Test



xsd:string **x1:** 3

```
999-99-9999
```

35.2 Using the XQuery Function Library in an XQuery

Adding an XQuery Function Library file to an XQuery.

Objectives

In this exercise, you will:

- Add the `protectSSN.xfl` file to an XQuery.
- Test the query.
- View the results.

Instructions

1. Build the DataServices project.
2. In the ALDSP-console, navigate to:

   `DataServices\CustomerManagement\CustomerProfile`
3. Click Admin and then Security.

4. Click the Access Policy icon for `getCustomerProfile()`.
5. Remove the users Bob and Joe from the Policy Statement list.
6. Test the `getCustomerProfile()` function without the protectSSN function by completing the following steps:
   a. Open `CustomerProfile.ds` in Test View.
   b. Select `getCustomerProfile()` from the function drop-down list.
   c. Enter CUSTOMER3 in the Parameter field.
   d. Confirm that the query returns a valid SSN.
7. Set SSN protection, by completing the following steps:
   a. Open `CustomerProfile.ds` in Source View.
   b. Expand the getAllCustomers node.
   c. Locate the SSN return code within the `getAllCustomers()` function. It should be as follows:

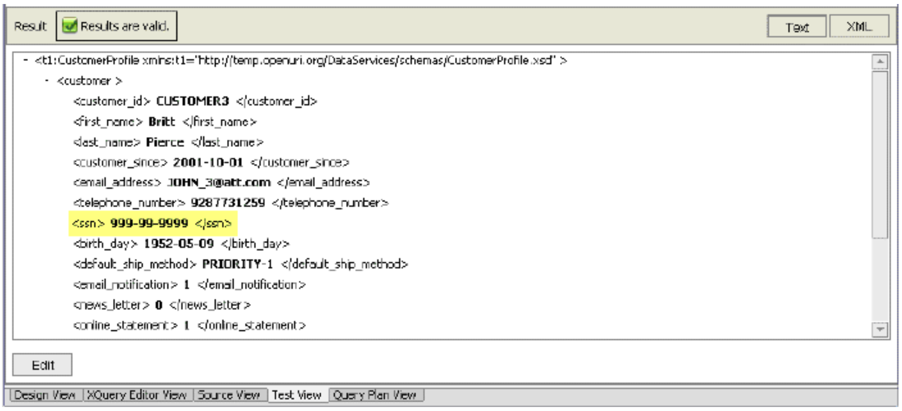
   ```
   <ssn?>{fn:data($CUSTOMER/SSN)}</ssn>
   ```

   d. In Data Services Palette, expand the xfl and library.xfl folders.
   e. Drag and drop `protectSSN()` to the SSN return value.
   f. Modify the remaining code, so that it is as follows:

   ```
   {ssn?{ns9:protectSSN($CUSTOMER/SSN)}</ssn>)}
   ```

   Note: <a></a> is the renamed element. You can use any name for the element, but for the sake of clarity, we used the simple <a></a> name.
8. Test the `getCustomerProfile()` function with the protectSSN function, by completing the following steps:
   a. Open `CustomerProfile.ds` in Test View.
   b. Select `getCustomerProfile()` from the function drop-down list.
   c. Enter CUSTOMER3 in the parameter field. The query should return an invalid social security number.

Figure 35-5 Test View of Protected SSN



```
- <t1:CustomerProfile xmlns:t1="http://temp.openuri.org/DataServices/schemas/CustomerProfile.xsd" >
  - <customer >
      <customer_id> CUSTOMER3 </customer_id>
      <first_name> Britt </first_name>
      <last_name> Pierce </last_name>
      <customer_since> 2001-10-01 </customer_since>
      <email_address> JOHN_3@att.com </email_address>
      <telephone_number> 9287731259 </telephone_number>
      <ssn> 999-99-9999 </ssn>
      <birth_day> 1952-05-09 </birth_day>
      <default_ship_method> PRIORITY-1 </default_ship_method>
      <email_notification> 1 </email_notification>
      <news_letter> 0 </news_letter>
      <online_statement> 1 </online_statement>
```

Lesson Summary

In this lesson, you learned how to:

- Create a XFL function.
- Use the XFL function within a query.

Glossary

ad-hoc query. A hand-coded or generated query that is passes to Data Services Platform on the fly, rather than stored in the ALDSP repository.

administration console. A Web-based administration tool that an administrator uses to configure and monitor WebLogic Servers. ALDSP provides a console to help manage instances of Data Services Platform.

application. A collection of all resources and components deployed as a unit to an instance of WebLogic Server. The application contains one or more projects, which in turn contain the folders and files that make up your application. Only one application can be open at a time.

cache. The location where ALDSP stores information about commonly executed stored queries for subsequent, efficient retrieval, thereby enhancing overall system performance. ALDSP provides query plan cache and result set cache.

cache policy. In the result set cache, configuration settings determine when the cached results expire for individual stored queries.

data model. A visual representation of data resources.

data object. In SDO, a complex type that holds atomic values and references to other data objects.

data service. A modeled object that describes a data shape and functions used to retrieve and update the data, as well as functions to navigate to other related data services.

data service mediator. The SDO mediator that uses data services to retrieve and update data.

data service update. The engine responsible for handling submits of changes to SDOs

data source. Any structured, semi-structured, or unstructured information that can be queried. The types of data sources that ALDSP can query include relational databases, Web services, flat files (delimited and fixed width), XML files, Java functions, application views using Web applications (business-level interfaces to the data in packaged applications such as Oracle, PeopleSoft, or SAP), data views (dynamic results of ALDSP queries).

data source schema. An XML schema that defines the content, semantics, and physical structure of a data source.

function. A uniquely named portion of an XQuery that performs a specific action. In the case of ALDSP the function would typically query physical or logical data.

Java Server Page (JSP). A J2EE component that extends the Servlet class, and allows for rapid server-side development of HTML interfaces that can be co-mingled with Java.

logical data service. A data service that integrates data from multiple physical and/or logical data services.

mapping. The process of connecting data source schemas to a target (result) schema.

metadata. Descriptors about a data service's information, format, meaning, and lineage.

physical data service. The leaf-level data services that expose external data. For relational sources, this would be a data service representing tables or stored procedures. For functional sources, this would be the functions that are considered to be the initial source of data operated on by XQuery.

project. Groups related files within an application.

query. In ALDSP an XQuery function that retrieves data from a data source. Functions define what tasks the query will perform, while expressions define what data to extract.

query operation. Operation that a query performs, such as a join, aggregation, union, or minus.

query plan. A compiled query. Before a query is run, ALDSP compiles the XQuery code into an executable query plan. When the query executes, the query plan is sent to the data source for processing.

repository. File-based metadata maintained in a ALDSP project.

result set. The data returned from an executed query. There are two types of result sets: intermediate result sets are temporary result sets that the query processor generates while processing an analytical query; final result sets are returned to the client application that requested the query in the form of XML data.

return type. A type of XML schema that defines the shape of data returned by a query.

schema. A model for representing the data types, structure, and relationships of data sets and queries.

security. Set of mechanisms available to prevent access to, corruption of, or theft of data. ALDSP extends the WebLogic Server compatibility security mechanisms to define groups, users, and access control to ALDSP resources.

service data object (SDO). Defines a Java-based programming architecture and API for data access.

Simple Object Access Protocol (SOAP). An extensible, platform-independent, XML-based protocol that allows disparate applications to exchange messages over the Web. SOAP can be used to invoke methods on servers, Web services, application components, and objects in a distributed, heterogeneous environment. SOAP-based Web services are one of the data sources ALDSP supports.

source schema. XML schema that describes the shape (structure and legal elements) of the source data—that is, the data to be queried. The ALDSP-enabled server runs queries against source data and returns query results in the form of the source schema.

stored query. A query that has been saved to the ALDSP repository. There is a performance benefit to using a stored query because its query plan is always cached in memory, optionally along with query result. With an ad-hoc query, however, the query plan and result are not cached. In addition, caching of query results for a stored query is configurable through the Cache tab on the ALDSP node in the Administration Console.

Structured Query Language (SQL). The standard, structured language used for communicating with relational databases. Database programmers use SQL queries to retrieve information and modify information in relational databases. In order to be able to access different types of data sources dynamically, ALDSP employs the XML-based XQuery language as a layer on top of platform-dependent query systems such as SQL.

target schema. See return type.

Weblogic Server. The platform upon which ALDSP is built.

Weblogic Workshop. The IDE in which ALDSP runs as an application.

Web service. Business functionality made available by one company, usually through an Internet connection, for use by another company or software program. Web services are a type of service that can be shared by, and used as components of, distributed Web-based applications. Web services communicate with clients (both end-user applications and other Web services) through XML messages that are transmitted by standard Internet protocols, such as HTTP. Web services endorse standards-based distributed computing. Currently, popular Web Service standards are Simple Object Access Protocol (SOAP), Web services description language (WSDL), and Universal Description, Discovery, and Integration (UDDI).

Web services description language (WSDL). Specification for an XML-based grammar that defines and describes a Web service. A WSDL is necessary if two different online systems need to communicate without human intervention.

xml schema. A structured model for describing the structure, content, and semantics of XML documents based on custom rules. Unlike DTDs, XML schemas are written in XML data syntax and provide more support for standard data types and other data-specific features. When metadata about a data source is obtained, it is stored in an XML schema in the ALDSP repository.

XQuery. An XML query language, which represents a query as an expression which is used to query relational, semi-structured, and structured data.

xsd. An abbreviation for XML Schema Definition. An XSD file describes the contents, semantics, and structure of data within an XML document.