BEA AquaLogic® Data Services Platform

Retail Dataspace Sample Application Guide

Version: 3.0
Revised: April 2008

Contents:

Retail Dataspace Sample Application

This chapter provides a brief overview of the Retail Dataspace Sample Application (RTLApp), which is included with a complete AquaLogic Data Services Platform (ALDSP) installation, and also describes a scenario about how this sample application was developed by a fictitious enterprise named Avitek Ltd. The purpose of this scenario is to illustrate how ALDSP can aggregate data from highly disparate data sources, allowing access to that data through a single point of access that itself is easily integrated with other applications.

The following sections are included:

These sections are preceded by a brief description of a fictitious enterprise named Avitek Ltd.

About Avitek Ltd.

Avitek is a mythical retailer that has grown through acquisitions. The company started out selling apparel but recently merged with an electronic retailer to expand business and consolidate cost centers such as accounting and web development. Because of this acquisition the company now has two very different order management systems (OMS), one each for electronics and apparel orders. Avitek also has a customer relationship management (CRM) system to manage customer profile information. Finally, Avitek has a Customer Service system to manage the support cases for Electronic products.

General IT Goals

General immediate and mid-range goals for the newly consolidated IT department include:

- An easy solution to providing integration of different back-end resources.
- An easy solution to providing read/update to these back-end resources.
- An abstraction layer to hide the complexities of interacting with different systems.

Specific Projects

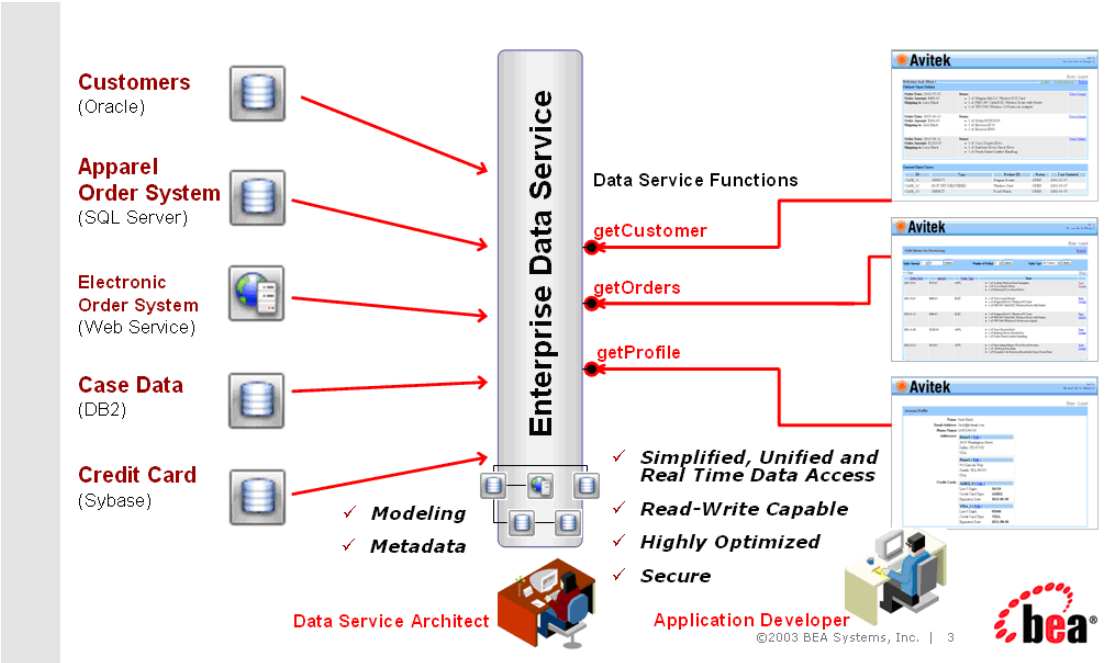
An early “low-hanging fruit” project is to build a customer self-service web site which can also be used by support representatives. This web site needs to provide an easy solution to integrating different back-end resources and to providing two-way communication (read/update). It will also need to provide an abstraction layer to hide the complexities of the different systems.

The front-end architecture of the Web site will provide a Java-based Customer Self-Service portal where customers can update their profile information and review or modify their orders. The back-end architecture will access the following five different sources of data (as shown in Figure 1):

- Customer related information stored in a CRM database (Oracle)
- Credit card information from a billing system (Sybase)
- Case information from a support database (DB2)
- Apparel order information from the company’s Order Management system (SQL Server)
- Electronics Order System accessed through a web service. (The original source of information is mainframe inherited with the merger.)

The project also needs a code name. RTLApp is selected.

Figure 1 RTLApp Front and Back End Architecture



Dataspace Projects in the Retail Dataspace Sample Application

Retail Dataspace Sample Application (RTLApp) includes the following dataspace projects:

- ElectronicsWS

  Contains a set of data services that form the basis for electronics equipment orders. A web service has been created from this dataspace project that serves as a data source for the RetailDataspace project. (The purpose of the Electronics dataspace project is to show how a web service can be used as a data source by another project.)

- RetailDataspace

  Main dataspace project demonstrated by RTLApp. The RetailDataspace project contains data services based on the following data sources:

  – The ElectronicsWS web service

  – The following relational data sources:

    • Order Management System, which contains the ApparelDB database

    • Billing Information, which contains the BillingDB database

    • Customer Relationship Management (CRM) System, which contains the CustomerDB database

    • Customer Service, which contains the CustomerDB database

The Retail Dataspace Sample Application also includes a separate client application, RTLSelfService, which demonstrates an application that invokes the RTLApp data services to display data in a Web application. Building, deploying and running the RTLSelfService application is described in "Quick Start Instructions for the RTLSelfService Application" on page 34.

Note: The source files and other application data for RTLApp are created and laid down in your domain directory tree by the wizards that you execute as part of the steps described in the sections that follow. These files do not exist on your machine prior to completing these steps.

Configuring the RTLApp Example and its Web Application

The following table provides links to topics which describe how to build and deploy the Retail Sample example under ALDSP versions 3.0, 3.2, and 3.01.

Links to instructions for running the RTLApp sample Web application are also provided.

Table 1 Version-specific Links for Configuring RTLApp and Installing its Sample Web Application

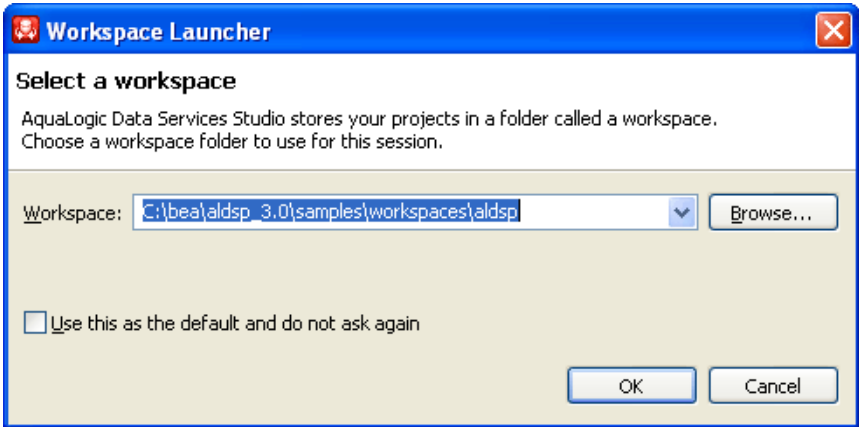
ALDSP 3.0	ALDSP 3.2	ALDSP 3.01
Build and deploy RTLApp	Build and deploy RTLApp and run the web-based sample	Build and deploy RTLApp
Install the sample retail application		Install the sample retail application

Note: If you are using ALDSP version 3.2, please see "*How to Configure the Retail Dataspace Sample Application for ALDSP 3.2*." This topic includes simplified instructions for creating and running the retail dataspace (RTLApp) sample web application.

To build and deploy the RTLApp dataspace projects, complete the following steps. If you have already built and deployed the dataspace projects in Data Services Studio, you can skip to "Importing the RetailDataspace Resources" on page 34.

1. Start Data Services Studio by choosing Start → All Programs → BEA Products → BEA AquaLogic Data Services Platform 3.0 → Data Services Studio.
2. In the Workspace Launcher, make sure that the Workspace is set to `<aldsp_home>/samples/workspaces/aldsp`, as in Figure 2, and click OK.

Figure 2 Data Services Studio Project Launcher



3. In the Welcome screen displayed in the Data Services Studio, click Retail Dataspace Sample, shown in Figure 3, which appears under the heading Install Sample Applications.

Figure 3 Link for Installing RTLApp

Data Services Studio displays a dialog box containing the message, "No server runtimes can be found in this workspace," and prompts you to click the Installed Runtimes... button.

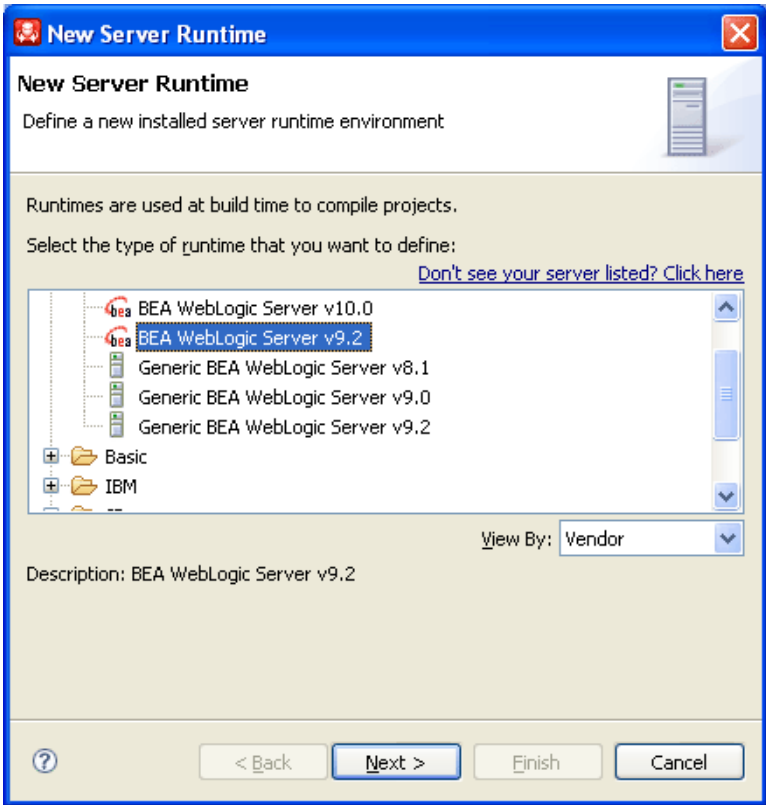
4. In the Application location section, make sure that Use default location is checked, and click Installed Runtimes...

   Data Services Studio displays a Preferences dialog box in which you add the home directory of the WebLogic Server 9.2 runtime that is contained in the same BEA Home directory as your ALDSP 3.0 installation.

5. In the Installed Server Runtime Environments dialog box, click Add...

   Data Services Studio displays the New Server Runtime dialog box, in which you select BEA WebLogic Server v9.2, as in Figure 4.

Figure 4 Server Runtime Environment Selection Dialog Box

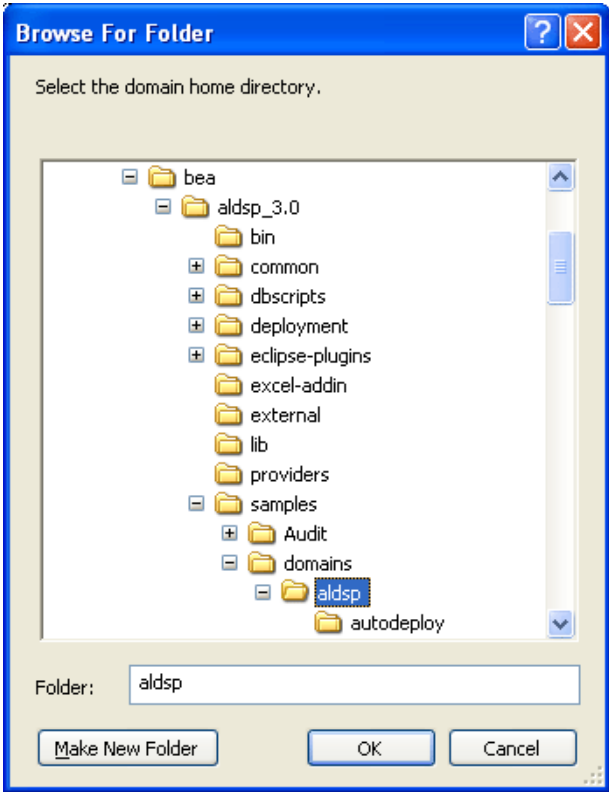


6. Click Next.

Data Services Studio then displays the Define a WebLogic Runtime dialog box, in which you specify the home directory of the WebLogic Server 9.2 installation associated with your ALDSP 3.0 installation.

7. Next to the field labeled WebLogic Home, click Browse... and select the WebLogic Server 9.2 home directory. By default, this directory name is weblogic92. Click OK.
8. Back in the Define a WebLogic Runtime dialog box, make sure the radio button for the JRE in your BEA Home directory is selected, and click Finish.
9. In the Installed Server Runtime Environments dialog box, click OK.
10. In the Retail Dataspace Sample — Server Configuration dialog box in which you specify the application location and WebLogic domain used for development, click Browse... next to the field labeled Domain home.
11. In the Browse for Folder dialog box, select the `<aldsp_home>/samples/domains/aldsp` directory, as in Figure 5, and click OK.

Figure 5 Samples Domain Home Directory Selection Dialog Box

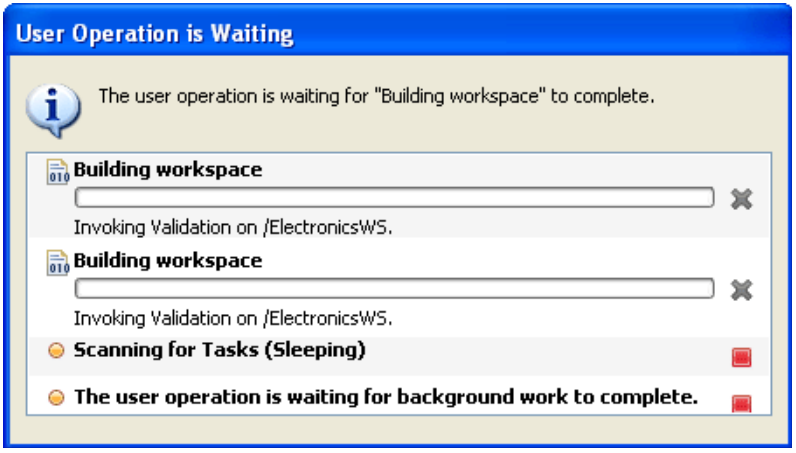


12. In the Retail Dataspace Sample — Server Configuration dialog box in which you specify the application location and WebLogic domain used for development, click Next.

13. In the Summary dialog box that is displayed, click Finish.

    Data Services Studio builds the sample dataspace, displaying a progress message box similar to Figure 6.

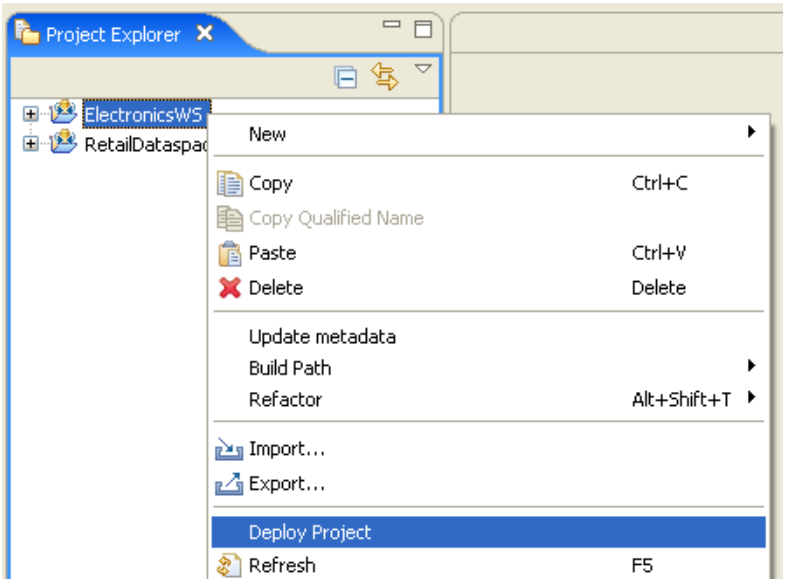
Figure 6 Dataspace Project Build Status Message Box



When the build process for the RTLApp dataspace projects is complete, the main Data Services Studio window is displayed.

14. Deploy the ElectronicsWS project by right-clicking the entry for it in the Project Explorer view, located on the left, and choosing Deploy Project, as in Figure 7.

Figure 7 Dataspace Project Deployment Menu Option



15. A message is displayed indicating that the server is not started, and prompts you to start it. Click Yes.
16. If you get a message indicating that the ElectronicsWS deployment is a success, click OK, and deploy the RetailDataspace project in the same way as the ElectronicsWS project.

The Challenge of Disparate Data

The sample retail application illustrates in simplified form the kinds of data integration challenges often encountered by Information Technology (IT) managers and staff. 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.
- Having such a single point of access to your data, can you develop reusable queries that are easily tested, stored, and retrieved?
- Once you can query data, can you as easily update it in a secure manner?
- Can you cache data to improve performance while protecting users from stale information?
- How difficult is it to consume query results in client applications?

Other questions may occur. Is the data-rich solution scalable? Is it reusable throughout the enterprise? Are the original data sources transparent to the application — or do they become an issue each time you want to make a minor adjustment? When changes to underlying data inevitably occur, how difficult will it be to propagate those changes to the application layer?

So many questions...

Business Case for the Avitek Self-Service Web Site

A survey commissioned by Avitek Marketing found customers to be dissatisfied with the call-in wait time required to track orders or update customer information. In a focus group the idea of a self-service web site resonated. Customer Service agreed; they have been requesting such a site for years, but the internal costs were always above budget. But now that Marketing is on board …

Web Site Design Requirements

Site requirements seem simple. Fulfillment identifies that customers will need to be able to:

- **Securely log in and out.**
- **Review order status.** A Home page will provide information on open orders, including product names and quantities of items ordered, order amount, shipping instructions, and a summary of any open customer service cases, with details.
- **Review order history.**
- **Review and change personal profile information.**
- **Search through orders.** A sample page provides the ability to retrieve orders based on a range of prices, range of dates, etc.

Bottom line: If customers can perform this level of self-service the company will save a lot of money.

Maintenance Requirements

A cross-sectional team of marketing and customer service develop maintenance requirements for the web application:

- 7x24 access
- Fewer than 10 HTML pages
- Low-maintenance
- Easily modified

Design Requirements

An application/UI designer begins “spec’ing out” the required JSPs. Pages are the easy part. Data is needed so he shoots off an email to the consolidated IT department.

Information Technology (IT) Weighs In: The Moment of Truth

When an IT data architect analyzes the requirements she turns up a problem. In surveying the information needed by the application — customer data for one data source, order data from two very separate divisions of the company (two more data sources), and customer support data (a fourth data source) — the architect realizes that integrating data from these diverse data sources

will be complicated and time consuming with additional maintenance problems down the road. Challenges included:

- **Cost of development.** Writing the code to access and integrate data from multiple diverse data sources would take more time than originally expected and require more expensive and scarcer resources.
- **Time to market**. Developing and testing an application against multiple data sources would extend beyond the date when the application is needed.
- **Cost of testing and maintenance.** High.

Perhaps most frustrating: little of the specialized code needed by the application can be reused.

So much for low-hanging fruit.

Search for an Alternative

Developing a unified view into distributed data is one of the most persistent challenges faced by IT departments. Just when you get all the available data sources normalized, new sources appear that must be dealt with, making yesterday's data integration solution practically obsolete.

This problem is so pervasive that each year thousands of arguably critically-needed applications go unwritten, are delayed, or are delivered in highly-compromised form because of the data integration challenges faced by even the most sophisticated enterprises.

Compared to the above, the RTLApp team preferred a solution that:

- Provides a layer of data abstraction so that queries can treat highly-divergent data sources as a single, virtual data source.
- Allows development of human-readable, reusable queries.
- Can be easily accessed by consuming applications through a simple API.
- Protects the integrity and security of the underlying data.
- Allows read-write testing and deployment.

A Possible Solution

When Avitek looked at AquaLogic Data Services Platform, they found a product that addressed the underlying challenges posed by the *apparently* simple RTLApp:

- Addressing the problem of data access, ALDSP provides data integration through a highly-accessible graphical interface.
- In ALDSP, queries are developed graphically and are self-describing.
- Once data integration is achieved, persistent queries are generated.
- Read-write to live or staged data is available at every step of the development process.
- Because ALDSP is based upon WebLogic Server, data caching and enterprise-level security is built in.

Specifically, the features that the team found most appealing included:

Data Access. ALDSP allows the application to access information from anywhere in the company — or beyond — through an easily-created *virtual data access layer*. Once accessed, data can easily be aggregated through a combination of reusable queries and views that are maintained in the ALDSP server.

Query Development. Then, once the data is collected under a single point of access, it is not difficult to create query functions that consolidate data from these disparate sources and present a common, reusable view ready for more specialized queries.

The declarative form of ALDSP artifacts (query functions in data services) makes them very readable.

Query Deployment. Once developed, queries are easily integrated into a client application thorough a variety of access methods such as the ALDSP Mediator API, the ALDSP control in Workshop, JDBC, or web service.

RTLApp Dataspace Projects

Several BEA technologies are exercised by the RTLApp dataspace projects:

- **Query Development.** Data services and queries that draw data from multiple data sources are created in Data Services Studio. ALDSP provides an extension of XQuery that can be used for data service development called XQuery Scripting Extension (XQSE), described in "Use of XQuery Statement Extension (XQSE)" on page 27.

For more information about XQSE, see the following:

- *BEA XQuery Scripting Extension*
- *How to Use the BEA XQuery Scripting Extensions*

- **Query Access.** Query functions are accessible through the Data Services Studio (Test view only), the ALDSP mediator API, JDBC, Web services, and the ALDSP control and can be tested within the Data Services Studio.
- **Client-side Development**. Data Services Studio provides a single environment for modeling, designing, testing, tuning, and deploying data services and application logic via facilities such as the following:
  - Query Mapper
  - Source Editor
  - Test Editor
  - Query Plan
  - Update Mapper

The following sections examine key artifacts of the RTLApp dataspace projects and describe examples of some of the query building features provided by ALDSP:

- "RTLApp Data Sources" on page 14
- "RTLApp Data Services" on page 15
- "Model Diagrams" on page 25
- "Building Queries" on page 26

RTLApp Data Sources

Although the RTLApp is very simple, the underlying data acquisition is challenging because data comes from four heterogeneous data sources. These are:

- **Customer Relationship Management (CRM) system.** CRM data (customer and credit card information) is stored in a database called RTLCUSTOMER.
- **Order Management System (OMS).** Avitek has two order management systems:
  - **Apparel products.** Information is maintained on site in a second database. This is represented in ALDSP as RTLAPPLOMS.

  - **Electronic products.** OMS information is available from a legacy system (RTLELECOMS) via a web service. The web service has several methods such as `getCustomerOrderByCustomerID()`, that takes a customer ID as input and returns a list of customer open order information through a web service.

- **Customer Service.** Service data is stored in a third database. The schema for this data is RTLSERVICE.
- **Billing information.** Credit card information is maintained in a separate, highly secure database, RTLBILLING.

RTLApp Data Services

The following sections describe work done by some of the RTLApp data services and shows how they are organized within a dataspace project. When you create a dataspace project, it is a best practice to construct data services in multiple layers, with the upper layers being increasingly abstract from the physical data sources on which they are based. The RTLapp data services were created in Data Services Studio and exist in the following layers:

- Physical data services

  Physical data services correspond to the physical data sources that are needed by the dataspace project.

- Normalized data services

  RTLApp adds this layer to present the dataspace project's data services in a more database-neutral manner that, for example, translates vendor-specific data types or create element names that are more readable or standardized.

- Logical data services

  RTLApp adds this layer to create discrete data services that combine other data services that draw upon multiple data sources and that expose data operations in the way in which clients will consume them.

- Application data services

  RTLApp adds this layer to provide the actual applications that are composed from data services defined in the preceding layers and to determine which operations on those data services are exposed to client applications.

Figure 8 shows the data services that make up the RetailDataspace dataspace project in RTLApp.

Figure 8 RetailDataspace Dataspace Project Data Services

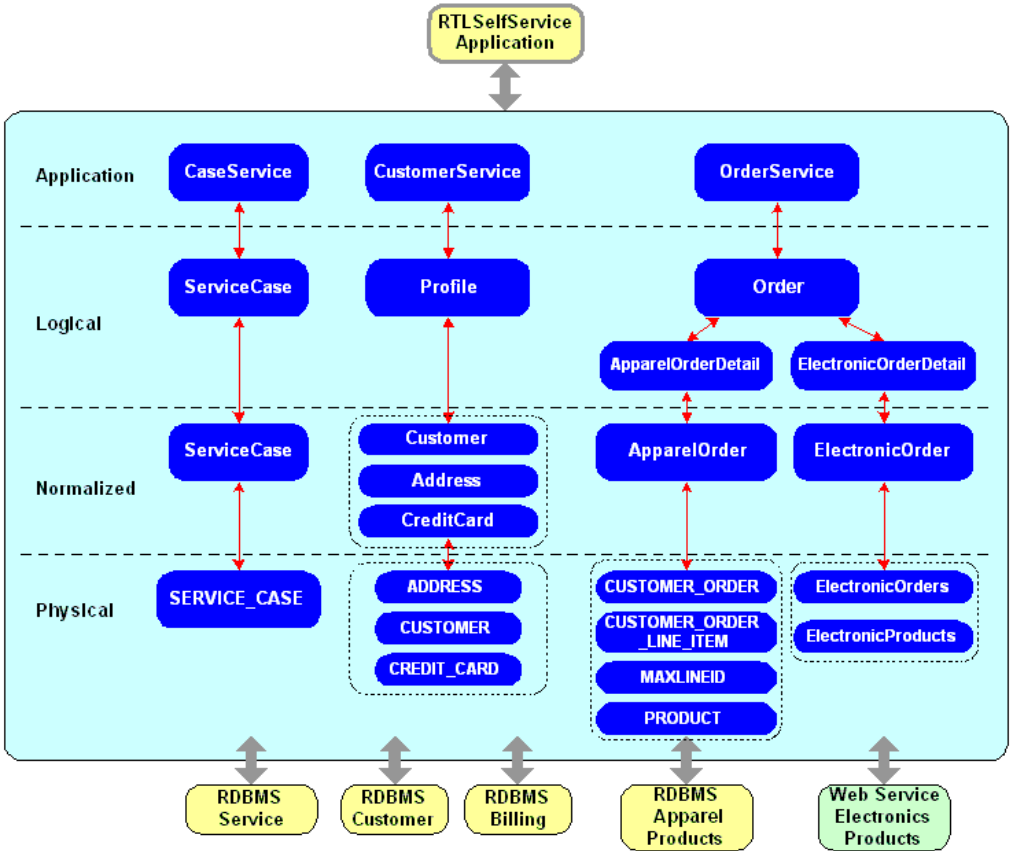
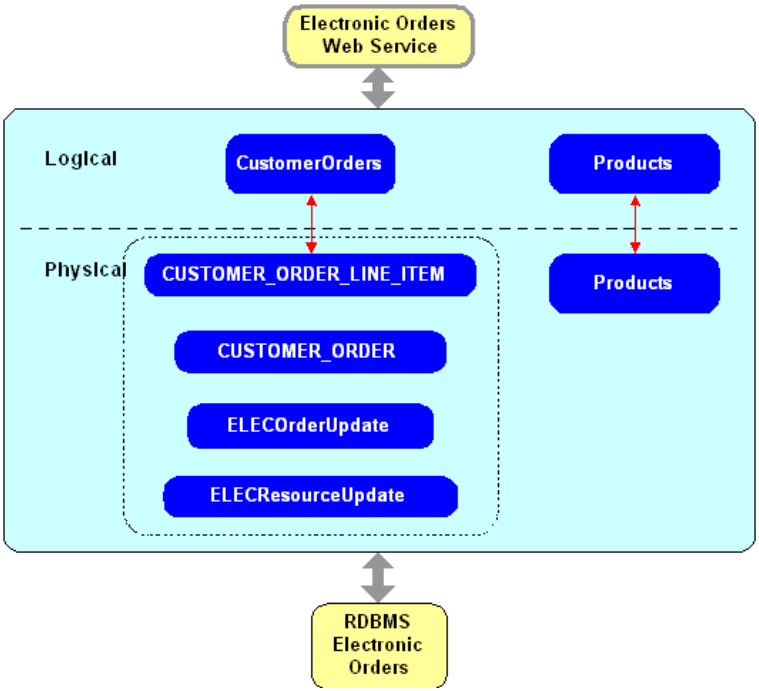


Figure 9 shows the data services that make up the ElectronicsWS dataspace project in RTLApp.

Figure 9 ElectronicsWS Dataspace Project Data Services



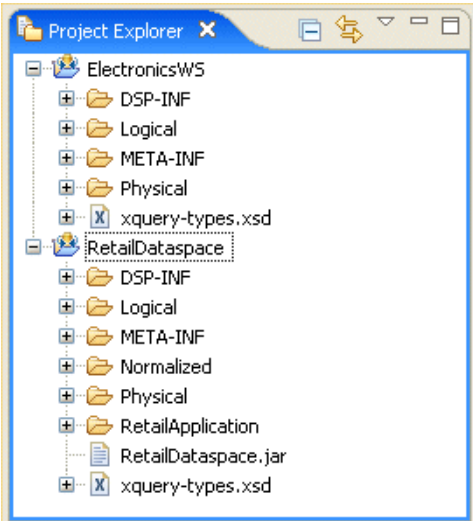
Viewing the Data Services in Data Services Studio

To open the RTLApp dataspace projects, start Data Services Studio, if not already running, by completing the following steps. These steps assume that you have already installed the RTLApp dataspace projects as described in "Dataspace Projects in the Retail Dataspace Sample Application" on page 3.

1. Start Data Services Studio by choosing Start → All Programs → BEA Products → BEA AquaLogic Data Services Platform 3.0 → Data Services Studio.
2. In the Workspace Launcher, make sure that the Workspace is set to `<aldsp_home>/samples/workspaces/aldsp` and click OK.

Data Services Studio is displayed on your machine, with the RTLApp dataspace project view appearing in the Project Explorer window, as shown in Figure 10.

Figure 10 Initial Retail Sample Application Dataspace Project View



Like all dataspace projects in Data Services Studio, components of the RTLApp sample are arranged in folders. This section briefly describes the key folders and their contents.

RetailDataspace is the main dataspace project in RTLApp. It encompasses all the data sources and data services that are accessed by client applications to RTLApp, and it shows a number of best practices for data service design and implementation. The ElectronicsWS dataspace project is included specifically to demonstrate the use of a web service as a data source and does not show best practices in the same way; however, a limited number of examples shown in this document are taken from ElectronicsWS as well.

The following key folders are described:

- Physical Data Services Folder
- Normalized Data Services Folder
- Logical Data Services Folder
- RetailApplication Data Services Folder

Note: Because the RetailDataspace dataspace project demonstrates best practices of data service design, most of the examples shown in this section are taken from that dataspace.

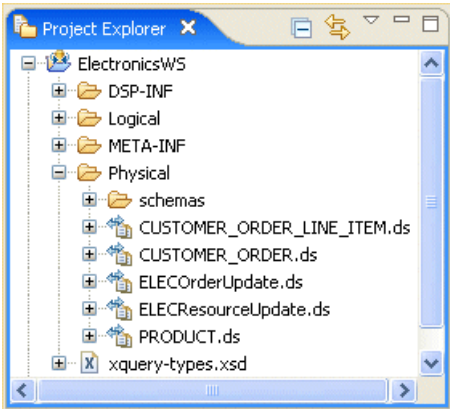
Physical Data Services Folder

Physical data services correspond to the physical data sources, described in "RTLApp Data Sources" on page 14, that are needed by the dataspace project. For example, the physical data services in the RetailDataspace project include apparel orders (ApparelDB), credit card information (BillingDB), Customer Relationship Management (CRM) information (CustomerDB), electronic orders (ElectronicsWS), and customer service information (ServiceDB).

In the RTLApp sample, physical data services typically have a read function and, often, one or several navigation functions that correlate to the primary key/foreign key relationship between relational sources.

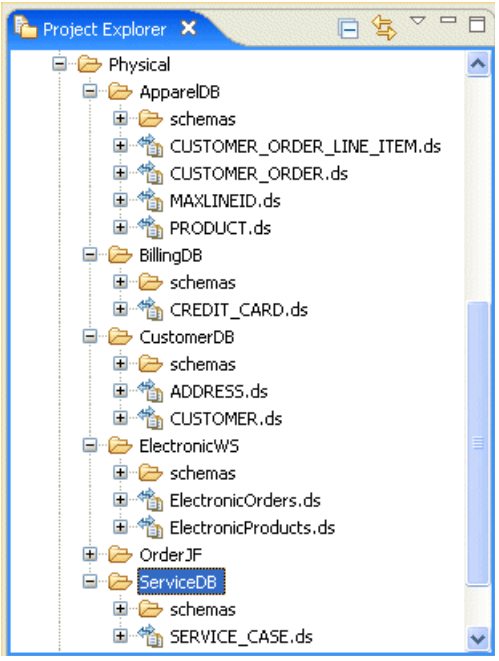
The folders, schema, and physical data services for the data sources included in the ElectronicsWS dataspace project are shown in Figure 12.

Figure 11 ElectronicsWS Physical Data Services



The folders, schema, and physical data services for the data sources included in the RetailDataspace project are shown in Figure 12.

Figure 12 RetailDataspace Physical Data Services



The physical data services in RTLApp are based on the data sources listed in Table 1.

Table 1 Data Sources for Physical Data Services in RTLApp

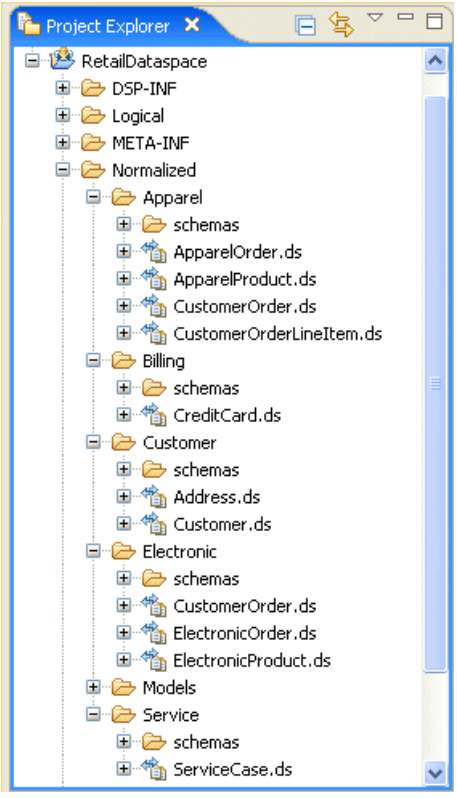
The data services in . . .	. . . are based on the following data sources . . .
ElectronicsWS/Physical folder: • CUSTOMER_ORDER_LINE_ITEM.ds • CUSTOMER_ORDER.pds • ELECOrderUpdate.ds • ELECResourceUpdates.ds • PRODUCT.ds	RTL Electronics Order Management System (RTLELECOMS), which contains the following relational database tables: • CUSTOMER_ORDER_LINE_ITEM • CUSTOMER_ORDER • PRODUCT
RetailDataspace/Physical/ApparelDB folder: • CUSTOMER_ORDER_LINE_ITEM.ds • CUSTOMER_ORDER.ds • MAXLINEIS.ds • PRODUCT.ds	RTL Order Management System (RTLAPPLOMS), which contains the ApparelDB RDBMS. Each of the following tables in ApparelDB is established as a discrete data source: • CUSTOMER_ORDER_LINE_ITEM • CUSTOMER_ORDER • PRODUCT
RetailDataspace/Physical/BillingDB folder: • CREDIT_CARD.ds	Billing Information System (RTLBILLING), which contains the BillingDB RDBMS. A data source is established for the BillingDB table CREDIT_CARD.
RetailDataspace/Physical/CustomerDB folder: • ADDRESS.ds • CUSTOMER.ds	Customer Relationship Management System (RTLCUSTOMER), which contains the CustomerDB RDBMS. Each of the following tables in CustomerDB is established as a discrete data source: • ADDRESS • CUSTOMER
RetailDataspace/Physical/ServiceDB folder: • SERVICE_CASE.ds	Customer Service (RTLSERVICE), which contains the ServiceDB RDBMS. A data source is established for the ServiceDB table SERVICE_CASE.
RetailDataspace/Physical/ElectronicWS folder: • ElectronicOrders.ds • ElectronicProducts.ds	ElectronicsWS web service, which serves as a data source for the physical data services for electronics orders

Normalized Data Services Folder

Normalized data services are located in a folder called Normalized, and they add a layer of abstraction or translation to the physical data services. At the normalized layer, the RTLApp data services are customized so that the data can be managed and used in a more database-neutral manner. Customizations may include translating vendor-specific data types on some fields and creating more readable, or more standardized, data element names. In addition, customized schema types, schema URLs, and the schema's target namespace may typically created in this layer. The mapping of the physical data elements to the normalized schema can be done using the XQuery Editor.

The folders, schema, and data services in the Normalized folder of the RetailDataspace project are shown in Figure 13.

Figure 13 RetailDataspace Normalized Data Services



Logical Data Services Folder

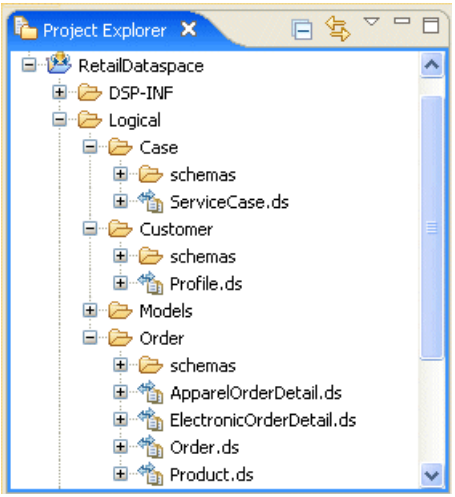
Logical data services (services based on normalized data services or other logical data services) for a dataspace project are located in a folder called Logical. It is in these logical data services that the read and navigation functions drawing on multiple data sources are developed and maintained.

Logical data services expose data operations in the way in which clients should consume them, that reflect how data services should interoperate with each other, and that are enriched with the business data vocabulary of the enterprise. For example, the RetailDataspace dataspace contains the logical data service `Profile.ds`, which combines the following data tables:

- The `ADDRESS` and `CUSTOMER` tables in CustomerDB
- The `CREDIT_CARD` table in BillingDB

The folders, schema, and data services in the Logical folder of the RetailDataspace project are shown in Figure 14.

Figure 14 RetailDataspace Logical Data Services



ALDSP also enables you to create a single logical data service that is drawn from disparate data sources. For example, the RetailDataspace dataspace contains a logical data service, `Order.ds`, that includes: `ApparelOrderDetail.ds`, which is based upon a relational data source; and `ElectronicOrderDetail.ds`, which is based upon a web service data source originating in a

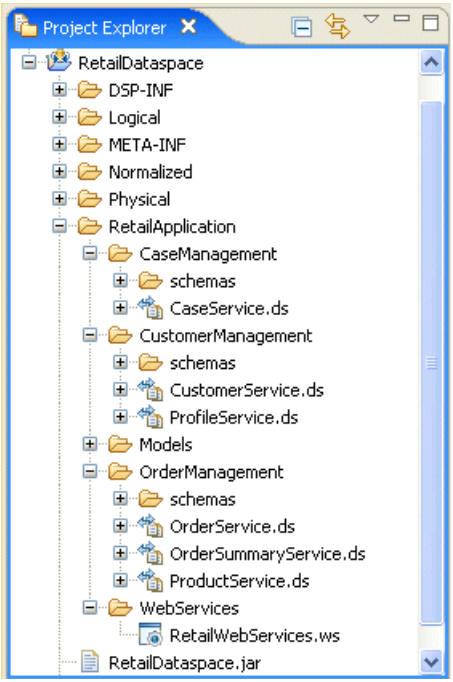
separate dataspace. The ALDSP feature that enables the union of two data sources is described further in “Query Union” on page 27.

RetailApplication Data Services Folder

The highest-level abstraction presented in RTLApp is the application layer. Users can define multiple applications composed from data services defined in the previous, or lower, layers.

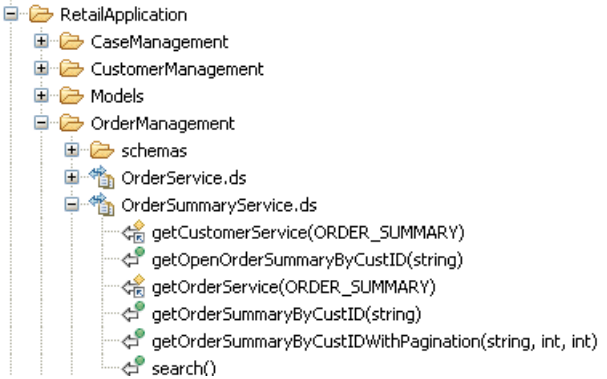
The folders, schema, and data services in the RetailApplication folder of the RetailDataspace project are shown in Figure 15.

Figure 15 RetailApplication Data Services



The operations of the RetailApplication data services that can be exposed to client applications can be identified graphically within Data Services Studio. For example, if you click the `OrderManagement/OrderSummaryService.ds` data service to expose its operations, you notice a small green ball icon next to several operations, shown in Figure 16.

Figure 16 Public Operations in OrderSummaryService Data Service



The green ball indicates that the following operations are publicly accessible by applications that invoke data services in the RetailApplication dataspace:

- `getOpenOrderSummaryByCustID`
- `getOrderSummaryByCustID`
- `getOrderSummaryByCustIDWithPagnation`
- `search`

The following operations are not publicly accessible:

- `getCustomerService`
- `getOrderService`

In the Workshop IDE, when the ALDSP control is generated for the RetailApplication data services, the client application sees only the public functions. When the Mediator Client JAR file is generated from the RetailApplication data services, only the public functions are contained.

Model Diagrams

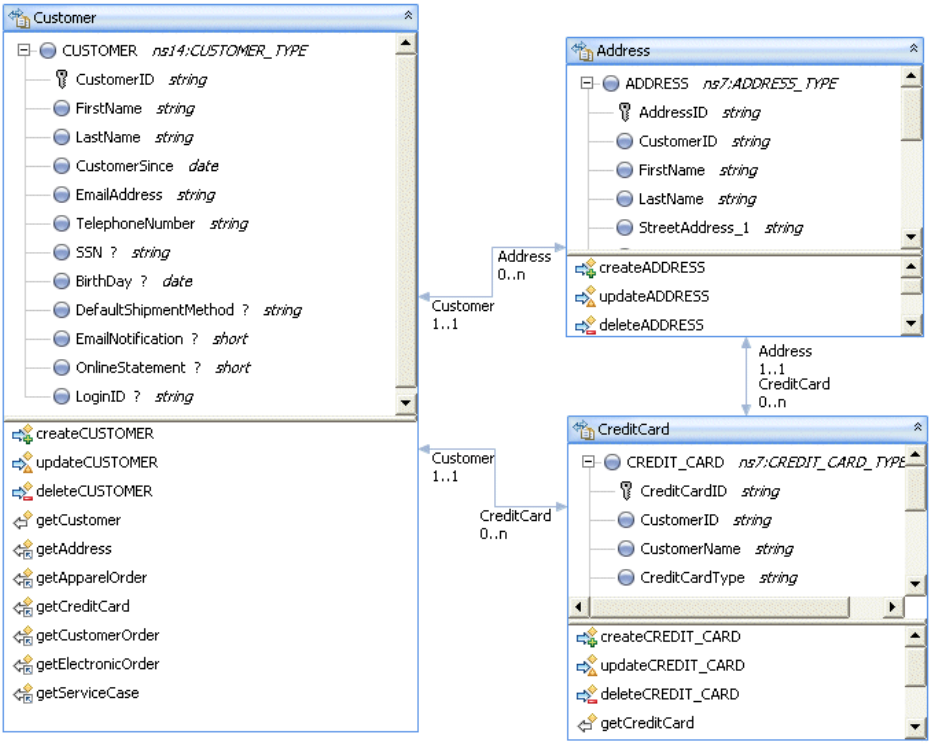
The Models folder of a logical data service folder contains model diagrams that are based on both physical and logical data services. Models are available in every main dataspace layer except in the physical layer. The purpose of models are two-fold:

- Models give the ALDSP user a big picture of the current layer in dataspace project. For example, the models in a given logical folder show the relationships among the data services in that folder.

- A convenient, graphical tool for establishing the functional relationship between any two data services that have something in common. Once you define the relationship function, ALDSP automatically generates the code represented by that function and incorporates it into the respective data services.

For example, the model `RetailDataspace/Normalized/Models/Customer.md`, shown in Figure 17, shows how an order is created: the customer has a relationship with a profile and with a product.

Figure 17 Customer Model



Building Queries

The following sections highlight examples of select features built into ALDSP that provide a simple and convenient way to enable powerful query capabilities inside of data services

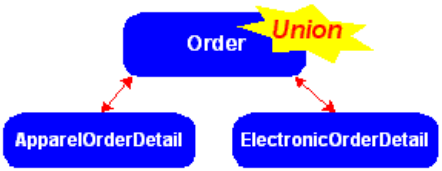
Query Union

The RetailDataspace project provides an example of a query union that concatenates a query for a relational data source to a query for a web service data source. ALDSP supports the ability to display this union in the Update Map, which can support the Create, Update, and Delete (CUD) operations automatically.

The RetailDataspace project uses this capability to combine the following product orders inside a single logical data service, `Order.ds`:

- An apparel order that originates from the relational database ApparelDB
- An electronics order that originates from the Electronics web service and also the order comes from the ElectronicsWS web service

Figure 18 Query Union



To display this query union, double click the `getOrderByCustID(string)` function of the data service `RetailDataspace/Logical/Order.ds` in the Project Explorer.

The following code is displayed in the Source tab. The **code in bold** performs the union:

```
declare function tns:getOrderByCustID($custID as xs:string) as element(ns1:ORDER)*{
    (:this union doesn't work for the update map
    for $APPAREL_ORDER in ns8:getApparelOrderByCustID($custID)
    return $APPAREL_ORDER
    ,
    for $ELECTRONIC_ORDER in ns9:getElectronicOrderByCustID($custID)
    return $ELECTRONIC_ORDER :)

    (:this union works for the update map:)
    for $ORDER1 in (ns8:getApparelOrderByCustID($custID),
         ns9:getElectronicOrderByCustID($custID))
    return $ORDER1
};
```

Use of XQuery Statement Extension (XQSE)

XQSE is a BEA extension to XQuery that adds procedural constructs — including basic statements, control flow, and user-defined procedures — to XQuery. Thus, XQSE is a superset

of XQuery, extending it with additional features that enable a much richer set of data services to be built without leaving the XML/XQuery world.

The following examples describe how XQSE is used in RTLApp.

Example 1: Using a Java Function to Update an Element

XQSE extends the base XQuery data model with information about elements that have been updated and then resubmitted to ALDSP. An XML node that contains such changes has the new XQSE type `changed-element`, which represents an element with changes.

In the RetailDataspace project, the normalized data service `ApparelOrder` contains a primary update operation, `updateAPPAREL_ORDER`. This operation passes a `changed-element` to the physical data service `ApplOrderUpdate`. `ApplOrderUpdate` is based on a Java function that updates the value of the element and returns it to the caller. The source of the Java function is included in `RetailDataspace\DSP-INF\lib\APPLOrderUOV.jar`.

Example 2: Implementing a Web Service Update

In the RetailDataspace project, the normalized data service `CustomerOrder` uses XQSE in the function `updateELEC_ORDER` to determine how to invoke the `CustomerOrders.ws` web service to update electronics orders. In addition, `CustomerOrder` is designed as a reflection of the ElectronicsWS logical data service `CustomerOrder`. This ALDSP capability allows RTLApp to eliminate the web service wrappers (*xxx*`Request` and *xxx*`Response`) that are defined by the WSDL when the ElectronicsWS `CustomerOrder` data service is exposed as a web service.

For an in-depth walkthrough of this example, see "Updating Data Based on a Web Service" on page 29.

Example 3: Implementing the ELEC Order Updates

In the RetailDataspace project, the normalized data service `CustomerOrder` contains a procedure, `updateELEC_ORDER`, which is implemented through a web service interface. As a result, because we do not want to transfer the data graph between the dataspace and web service protocol, the `changed-element` is not available in `CustomerOrders`. Therefore, RTLApp uses XQSE in the `changeCustomerOrders` function in the logical data service `CustomerOrders` (of the `ElectronicsWS` dataspace project) to implement the logic for updating the element representing electronics orders. This logic is as follows:

1. Delete the Order and LineItems.
2. Re-insert the updated values.

Updating Data Based on a Web Service

ALDSP provides a robust set of automated services to handle updates to data services that are based on relational data sources. However, automated update services are not available for data sources that are based on web services. For web service based data sources, ALDSP permits the use of XQSE procedures, which can greatly simplify the creation of custom code for those update tasks. RTLApp provides an example of using XQSE to update the customer orders that are managed by a web service. This example demonstrates a design pattern for isolating a web service resource from the higher-level data services in a dataspace project. The sections that follow provide a detailed walkthough of this example.

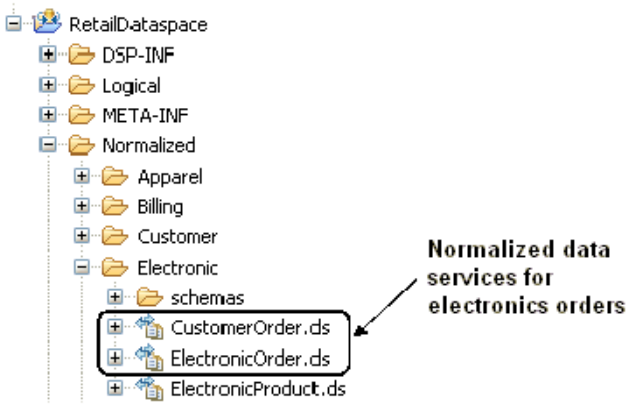
About Customer Orders for Electronic Products

As explained in "RTLApp Data Sources" on page 14, orders for electronics products are managed by the web service, CustomerOrders. This web service is created from the logical data service `CustomerOrders`, which is in the ElectronicsWS dataspace project. This web service is subsequently imported into the physical layer of the RetailDataspace project. The physical data service, `ElectronicOrders`, uses the imported CustomerOrders web service as a data source.

Note: The creation of a web service from a logical data service, and then importing that web service into the physical layer of another dataspace project, is not meant to be a design pattern. However, RTLApp uses this method for two reasons: first, to show that it can be done; and second, to take advantage of an easy-to-use ALDSP capability to generate a simple, ad hoc web service that can serve the purposes of the RTLApp dataspace sample.

Normalizing a data service enables consuming applications to use the data without regard for the inherent differences of its various underlying source representations. The RetailDataspace project contains two normalized data services, shown in Figure 19, that together abstract the process of using and updating data that is managed by the CustomerOrders web service.

Figure 19 RTLApp Data Services Based on a Web Service



The following data services, circled in Figure 19, are the focus of how XQSE can be used in a data service to update data managed by a web service:

- `ElectronicOrder`

  This data service contains normalized elements for processing electronics orders, exposing them to logical data services as simple create-update-delete procedures. The `ElectronicOrder` data service abstracts the elements and procedures contained in the `CustomerOrder` logical data service. When create-read-update-delete (CRUD) operations are invoked on the `ElectronicOrder` data service to process customer orders, the `ElectronicOrder` data service invokes the corresponding CRUD operations on the normalized `CustomerOrder` data service.

- `CustomerOrder`

  This is the normalized data service that is based on the physical data service `ElectronicOrders`, which is in turn based upon the CustomerOrders web service that has been imported into the physical layer of the RetailDataspace project. The `CustomerOrder` data service normalizes the `ElectronicOrders` physical data service by removing multiple web service wrapper layers that result when that physical data service is created. When the normalized `ElectronicOrder` data service invokes the `CustomerOrder` data service to process a customer order update, the `CustomerOrder` data service uses XQSE to update the value of the customer order in the CustomerOrders web service, as explained in "XQSE Procedure Walkthrough" on page 32.

  The `CustomerOrder` data service therefore functions as a bridge between the normalized `ElectronicOrder` data service and the physical `ElectronicOrders` data service.

Figure 20 shows the relationship among the data services in the RetailDataspace project and the CustomerOrders web service when the data managed by that web service is updated.

Figure 20 Relationship Among RetailDataspace Data Services and CustomerOrders Web Service

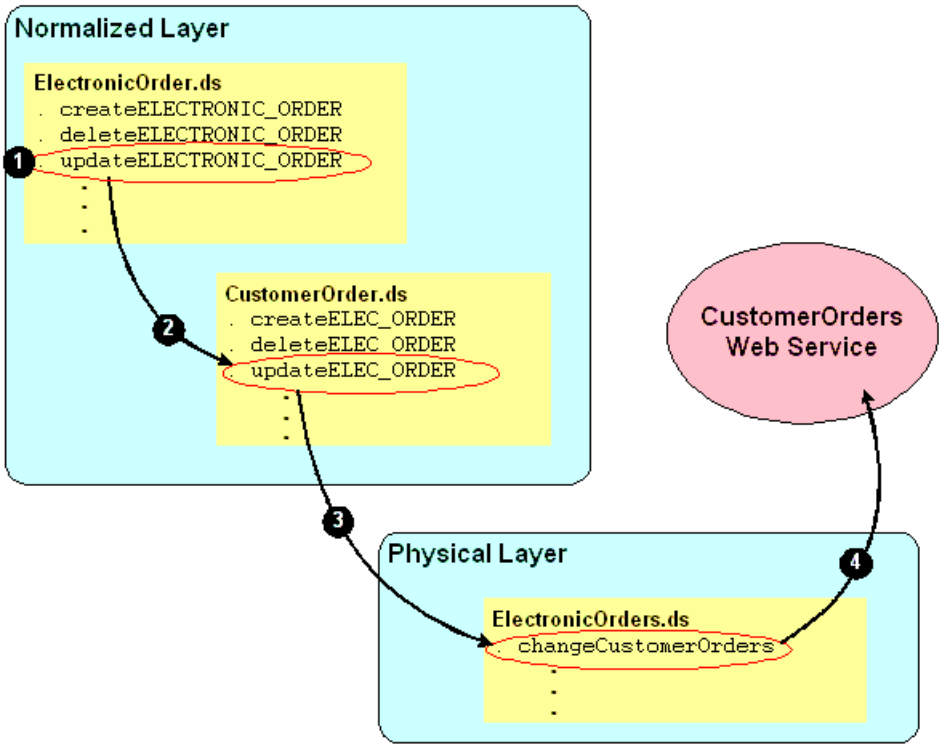


Figure 20 includes the following numbered callouts to highlight execution flow when the normalized data service `ElectronicOrder` is invoked to update the value of a customer order:

1. As the result of a client application that has submitted the values of a customer order, a logical data service in the RetailDataspace project invokes the `updateELECTRONIC_ORDER` operation.
2. The `ElectronicOrder` data service invokes the `updateELEC_ORDER` procedure of the `CustomerOrders` data service to effect the update of the value of the customer order.
3. The `updateELEC_ORDER` procedure invokes the `changeCustomerOrders` operation in the physical data service `ElectronicOrders` to construct a `changeCustomerOrders` object, which holds the value of the customer order.
4. The `changeCustomerOrders` operation of the physical data service `ElectronicOrders` invokes the CustomerOrders web service to submit the updated value of the customer order.

XQSE Procedure Walkthrough

The normalized `CustomerOrder` data service contains the XQSE procedure, `updateELEC_ORDER`, to perform the web service update, which cannot be accomplished via the automated update services available in ALDSP. Specifically, the `updateELEC_ORDER` procedure performs the following tasks:

1. Retrieves a new value for the customer order from the web browser client, RTLSelfService.
2. Creates a `changeCustomerOrders` SDO object and inserts the new value of the customer order into it.
3. Calls the `changeCustomerOrders` function in the physical data service `ElectronicOrders`, passing the `changeCustomerOrders` SDO object.

As a result of an invocation on the `updateELEC_ORDER` procedure, the CustomerOrders web service that has been imported into the physical layer of the RetailDataspace project invokes the logical data service `CustomerOrders` that is contained in the ElectronicsWS dataspace project. In turn, that `CustomerOrders` data service updates the electronics orders RDBMS, RTLELECOMS, which is described in "RTLApp Data Sources" on page 14.

Viewing the updateELEC_ORDER XQSE Procedure in Data Services Studio

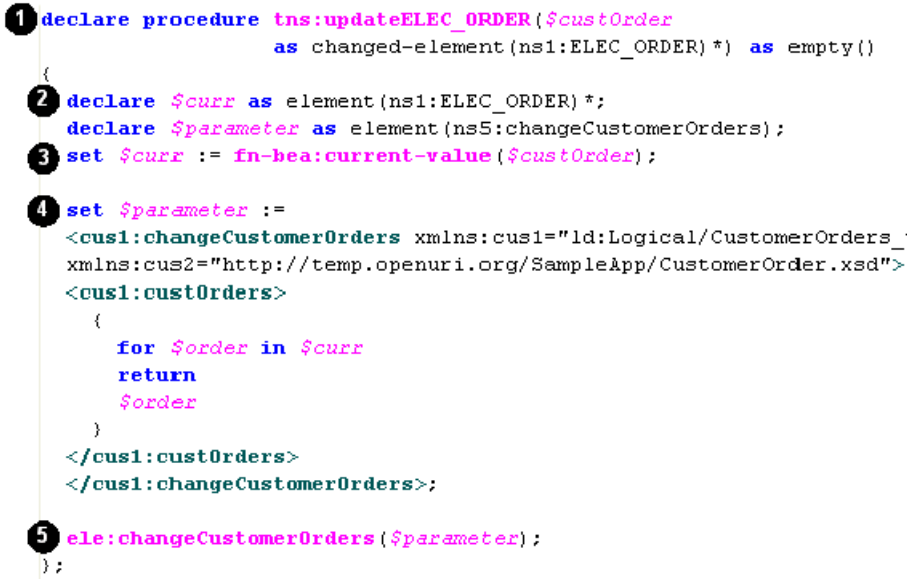
To view the `updateELEC_ORDER` XQSE procedure:

1. Start Data Services Studio, if necessary, as described in steps 1 and 2 in "Configuring the RTLApp Example and its Web Application" on page 4.
2. In the Project Explorer, double click the `CustomerOrder` data service, which appears in the Normalized/Electronic folder of the RetailDataspace dataspace project, as shown in Figure 19.
3. In the bottom of the workspace, select the source tab.
4. Expand the procedure named `updateELEC_ORDER`.

Description of XQSE Statements in updateELEC_ORDER

The `updateELEC_ORDER` procedure is shown in Figure 21.

Figure 21 XQSE Code for Updating a Web Service



For a description of the statements in the `updateELEC_ORDER` procedure, refer to the following callouts in Figure 21:

1. This XQSE procedure declaration establishes `updateELEC_ORDER` as the primary update function of the `CustomerOrder` data service. A primary update function must pass an argument that is declared to be of XQSE type `changed-element`, which contains the updates for the web service data source. The argument `$custOrder` is a data graph that simultaneously contains both the old and new values of the customer order.

   For more information about XQSE procedure declarations and the `changed-element` type, see BEA XQuery Scripting Extension (XQSE) in the *XQuery and XQSE Developer's Guide*.

2. Two variables are declared:

   – `$curr` — to represent the object containing the current values of the customer order, initially held by the web service

   – `$parameter` — to represent the SDO object that contains the new customer order, which is passed to the web service

3. This `set` statement invokes `fn-bea:current-value($custOrder)` on the `changed-element`, `$custOrder`. This statement causes `$custOrder` to be updated with the

new values of the customer order, which are assigned to the variable `$curr`. (The previous values of `$curr` are discarded.)

4. This `set` statement constructs a `changeCustomerOrders` SDO object, named `$parameter`, from the array of customer order values contained the `$curr` variable.
5. This statement invokes the `changeCustomerOrders` procedure in the physical data service `ElectronicOrders`, passing the `changeCustomerOrders` SDO object.

Quick Start Instructions for the RTLSelfService Application

RTLSelfService is a companion sample application provided with RTLApp that demonstrates the use of a Web-based client application that invokes data services and displays the resulting data in a Web browser.

The steps for configuring, building, deploying, and running the RTLSelfService application are provided in the following sections:

- "Importing the RetailDataspace Resources" on page 34
- "Building and Deploying the RTLSelfService Application in the Workshop IDE" on page 35
- "Running the RTLSelfService Application in a Browser" on page 39

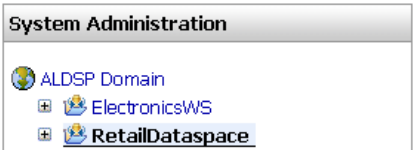
Importing the RetailDataspace Resources

The RTLApp sample application exercises ALDSP functionality that requires certain data service configurations, such as failover, caching, security, and others, to be set prior to building and deploying the RTLapp application in the Workshop IDE. These configurations are contained in the resource file, `RetailDataspace.jar`, which needs to be imported into the RetailDataspace project as explained in the steps that follow.

1. After the RetailDataspace project is deployed in Data Services Studio, start the AquaLogic Data Services Console by choosing Start → All Programs → BEA Products → BEA AquaLogic Data Services Platform 3.0 → Examples → AquaLogic Data Services Console.
2. Log in to the AquaLogic Data Services Console using the default username `weblogic` and password `weblogic`.

3. Select the System Administration category, and then select RetailDataspace from the navigation tree, shown in Figure 22.

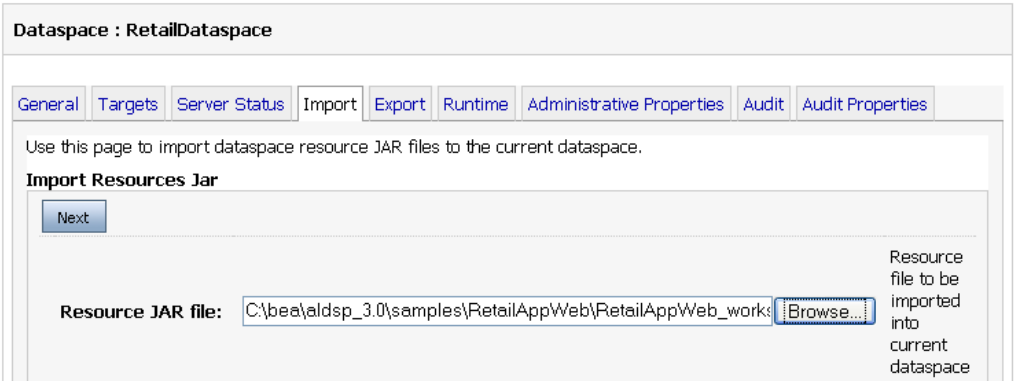
Figure 22 Dataspace Selection in ALDSP Administration Console



4. Click Lock & Edit.

5. Click the Import tab, then click Browse... to select the RetailDataspace JAR file `<aldsp_home>\samples\workspaces\aldsp\RetailDataspace\RetailDataspace.jar`, as in Figure 23.

Figure 23 Importing a Resource JAR



6. Make sure that the Full Deployment checkbox remains unchecked, and click Next.

7. Click Import.

8. Click Activate Changes.

Building and Deploying the RTLSelfService Application in the Workshop IDE

Note: If you are using ALDSP version 3.2, please see "*How to Configure the Retail Dataspace Sample Application for ALDSP 3.2*." This topic includes simplified instructions for creating and running the retail dataspace (RTLApp) sample web application.

To build and deploy the RTLSelfService application in the Workshop for WebLogic Platform 9.2 IDE, complete the following steps.

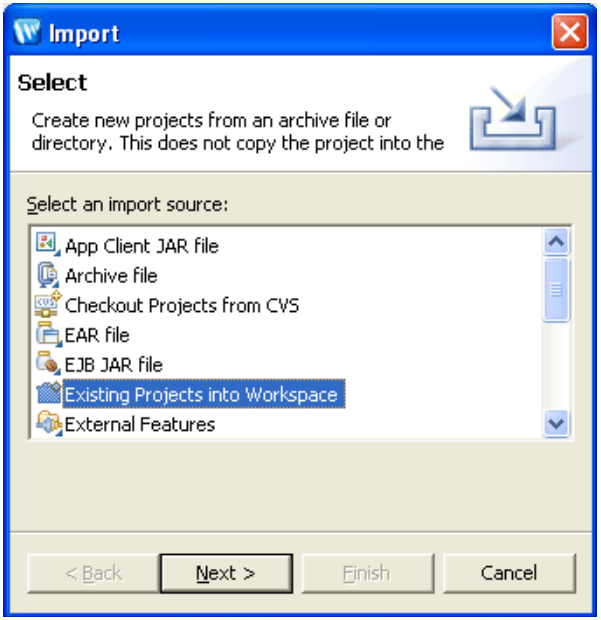
Note: The RTLApp sample can be run only in the ALDSP sample domain directory; that is, in the `<aldsp_home>/samples/domains/aldsp` directory.

1. Copy the file `com.bea.dsp.ide.control.feature.link` from the `<aldsp_home>/eclipse-plugins/workshop9` directory (for example, `C:\bea\aldsp_3.0\eclipse-plugins\workshop9`), and place it into the `<workshop92_home>/eclipse/links` directory (for example, `C:\bea\workshop92\eclipse\links`).
2. Start Workshop for WebLogic Platform 9.2 by choosing Start → All Programs → BEA Products → Workshop for WebLogic Platform.
3. In the Workspace Launcher dialog box, specify the workspace RTLAppWeb, as in the following example:

   ```
   C:\bea\user_projects\w4WP_workspaces\RTLAppWeb
   ```

4. In the Workshop IDE, choose File → Import..., and click Existing Projects into Workspace, as in Figure 24:

Figure 24 Project Archive Selection Dialog Box



5. Click Next.
6. In the Import Projects dialog box, click Select archive file, click Browse..., and select the following file:

   Windows:

   `<aldsp_home>\samples\RetailAppWeb\RetailAppWeb_workshop9.zip`

   Unix:

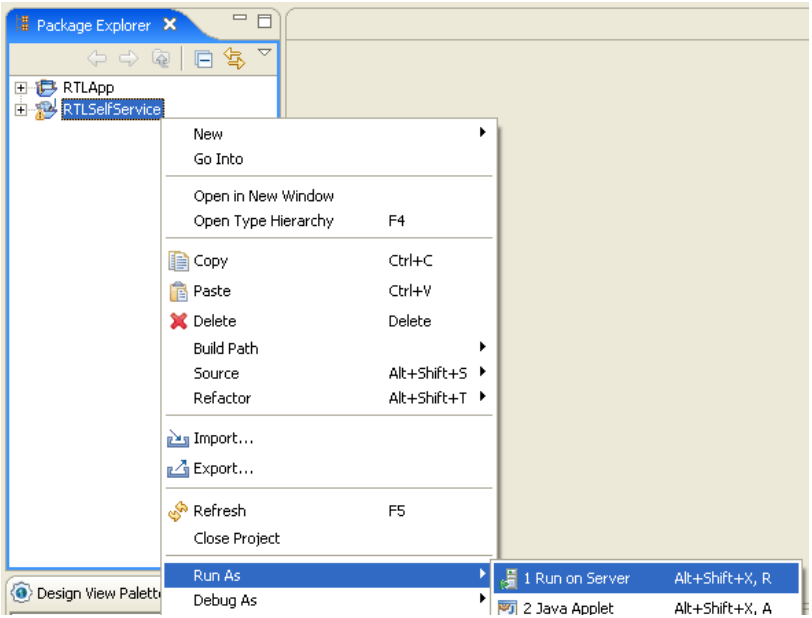
   `<aldsp_home>/samples/RetailAppWeb/RetailAppWeb_workshop9.tar`

   Note: The `<aldsp_home>/samples/RetailAppWeb` directory also contains the file `RetailAppWeb_workshop10` (`.zip` or `.tar`) because the Workshop for WebLogic Platform 10.0 IDE can also be used with ALDSP 3.0. For more information, see the *Release Notes*.
7. Click Open.
8. In the Import Projects dialog box, ensure that the RTLApp and RTLSelfService workspace projects are selected, and click Finish.

9. Allow a few minutes for the imported workspace projects to build, and ignore warning messages that may be displayed in the Servers view of the Workshop IDE.

10. In the Package Explorer view in the left, right-click the root node of the RTLSelfService workspace project, and choose Run As → Run on Server, as in Figure 25.

Figure 25 Workshop Menu Option for Running the RTLSelfService Application



11. In the Define a New Server dialog box, make sure the following are selected:

    – Server host name `localhost`

    – Server type BEA WebLogic Server v9.2 Server

    Typically you can accept the defaults, but you can click Installed Runtimes... to ensure that the server runtime is set to the WebLogic Server 9.2 runtime with which your ALDSP installation is associated.

12. Click Next.

13. In the Run on Server dialog box, click Browse... next to the field labeled Domain home, and select the following directory:

    `<aldsp_home>/samples/domains/aldsp`

14. Ensure the Use default checkbox below the label Name is unchecked, and click Finish.
15. After the startup of the RTLSelfService application is complete, the welcome screen is displayed in the Workshop IDE, as in Figure 26.

Figure 26 RTLSelfService Application Login Page

Running the RTLSelfService Application in a Browser

If you have not already done so, start the RTLSelfService Application by completing the steps in "Building and Deploying the RTLSelfService Application in the Workshop IDE" on page 35. When the login page shown in Figure 26 displayed, you can choose from users Steve, Jack, Tim, Homer, or Jerry. Each uses the password `weblogic`.

Once a customer logs in, she or he sees their MyProfile screen. From there he or she can navigate to information on open orders, order history, support, search, and logout. Customers can edit open orders (see Figure 28) and get details on completed orders. Search allows the user to supply product description information, start or end date, or order amount brackets.

The application also demonstrates some ALDSP facilities including the ability to:

- Refresh data
- Restrict or unrestrict access
- Make a data source unavailable
- Enable Cache
- Show SQL reports created from Crystal Reports
- Update data

My Profile Page

The My Profile page illustrates ALDSP's ability to perform automatic read/write on distributed data.

Figure 27 My Profile Page

My Profile page consolidates Customer and Credit Card information. Users can change their personal profile information, address information, and credit card information. Changes are initially reflected on their MyProfile page. If satisfied, the user can click Submit All Changes to persist changes to the respective data sources.

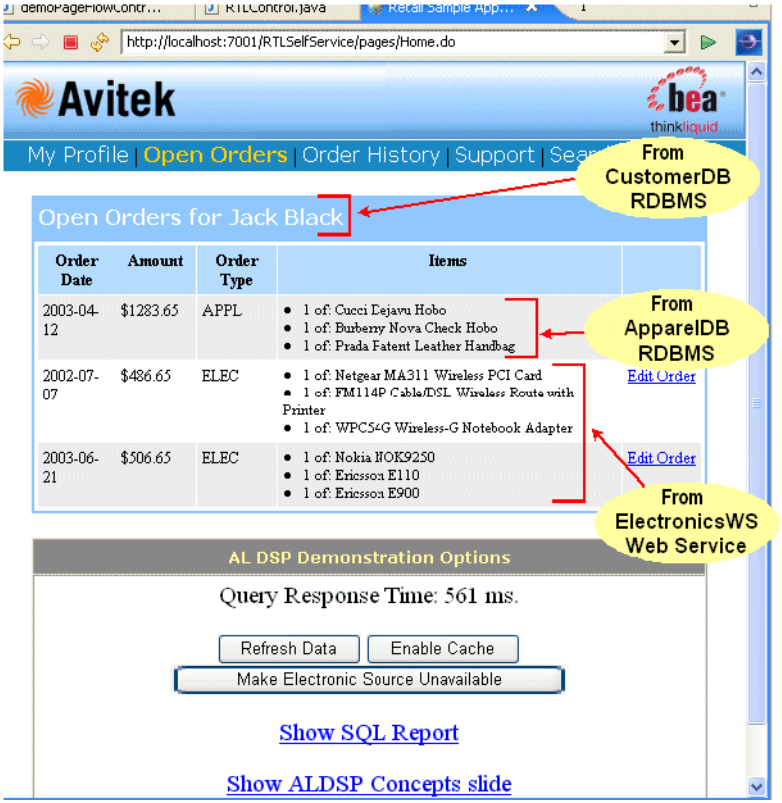
Page Design

The page is based on `ProfileView.jsp` and the ProfileView data service (`ProfileService.ds`).

Open Order Page

The Open Order page consolidates a customer's electronics and apparel open orders.

Figure 28 Open Order Page



Data Sources

From the user's perspective, changing data is simply a matter of select and type. However, the underlying update mechanisms for electronic orders (which are maintained as a web service) and apparel orders are quite different.

Update Mechanisms

Electronic orders are derived from a web service. In this case, updating a electronic order demonstrates web service custom update capabilities.

Apparel orders are derived from a relational database and updates are automatic.

Caching Options

The Enable Cache option turns on caching for the function underlying the Open Orders page. You can use the Refresh button to verify that the execution time when cache is enabled is significantly faster than when it is not.

Handling Unavailable Sources

Click the Make Electronics Source Unavailable button to disable the web service. This action also refreshes your Open Orders page. Notice that you can still retrieve partial results (apparel orders) when a data source becomes unavailable.

Access LD via JDBC

The Show SQL Report button illustrates accessing data through JDBC. This demonstrates the integration of ALDSP query functions with reporting and business intelligence tools such as Crystal Reports.

Page Design

The Open Order page is controlled by a JSP named `defaultView.jsp`. Call-outs in Figure 5 show underlying data sources. The page derives its font and other look-and-feel characteristics from a cascading stylesheet.

Order History Page

The Order History page displays historical order information for electronic and apparel orders.

Figure 29 Order History Page

Users have several options associated with viewing their previous orders:

- Items can be ordered by date, amount, or order type (electronic or apparel).
- A operational filter can be applied based on order amount.

- The user can restrict the number of orders retrieved to a pre-set amount (5, 10, 15, 20, or all).
- The user can restrict orders to apparel or electronic.

An Apply button is provided to control page refresh.

Security

ALDSP security can be demonstrated when the number of orders is set to All. Then when the Restrict Access option is selected, data is automatically *redacted* to show only orders where order amounts are less than $500.00.

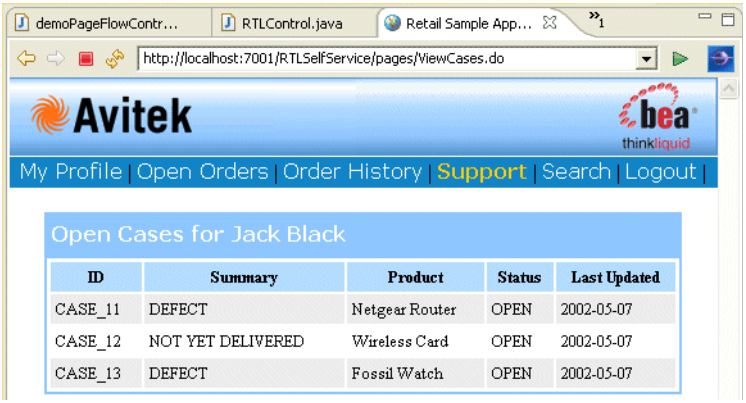
Page Design

The Order History page is controlled by a `OrderHistory.jsp`.

Support Page

The Support page provides information on any open support cases for the currently logged-in customer.

Figure 30 Customer Support Page

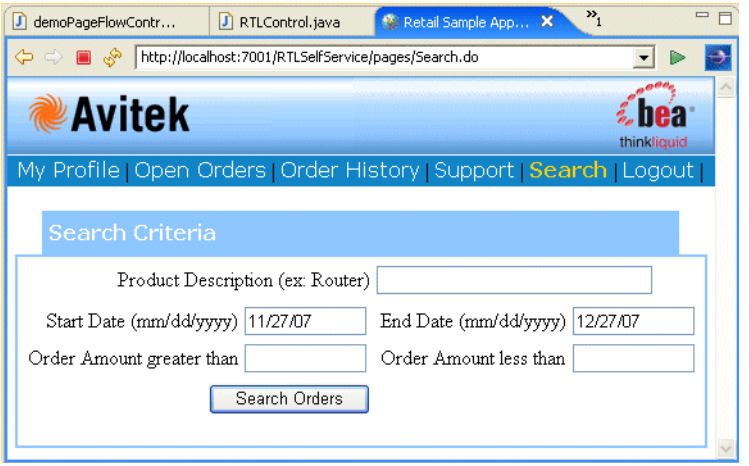


The underlying JSP is `caseView.jsp`.

Search Page

Users can search for specific orders based on order dates, items, or amounts. The underlying code executes ad hoc ALDSP query functions.

Figure 31 Search Page



Users can search on any combination of search field criteria including product description, start or end data, and a range of order amounts.

The JSP for this page initiates a small Java program that incorporates customer input and generates an XQuery based on the selected parameters.

Summary

In summary, the RTLApp provides:

- A virtual data access layer that allows you to treat heterogeneous data as from a single source.
- Ability to access the data through declarative queries that can be created in the Data View Builder or developed externally.
- Availability of ALDSP queries and server for easy integration into applications or processes.