BEA Liquid Data for WebLogic™

Liquid Data by Example

Version 8.1
Document Date: December 2003
Revised: January 2004

Copyright

Restricted Rights Legend

Trademarks or Service Marks

Contents

About This Document

1. Understanding the Avitek Customer Self-Service Sample Application

2. Query Cookbook

3. Samples Installed with Liquid Data

Index

About This Document

Read this document to learn how to build and test queries in XQuery language that can retrieve real-time information from heterogeneous data sources using the BEA Liquid Data for WebLogic server.

This document describes how to use the Data View Builder to design and generate XQueries with the Builder drag-and-drop tools, functions, source and target schemas. The focus of this document is on how to use the Data View Builder to create queries in Liquid Data. Liquid Data accepts queries written in XQuery, which is an Extensible Markup Language (XML) Query language that adheres to the standards described by the World Wide Web Consortium (W3C). The XQuery standard, version 1.0, is the structured query language used by the Liquid Data server.

This document covers the following topics:

- Chapter 1, "Understanding the Avitek Customer Self-Service Sample Application," describes the make-up of a sample application that uses
- Chapter 2, "Query Cookbook," introduces key concepts such as XQuery, ad hoc queries, and Builder-generated queries.
- Chapter 3, "Samples Installed with Liquid Data," provides detailed examples about how to construct queries using some advanced techniques and functions.

What You Need to Know

Users creating queries with Data View Builder should have an understanding of XML and XML schemas.

e-docs Web Site

BEA product documentation is available on the BEA corporate Web site. From the BEA Home page, click on Product Documentation or go directly to the "e-docs" Product Documentation page at e-docs.bea.com.

How to Print the Document

You can print a copy of this document from a Web browser, one file at a time, by using the File—>Print option on your Web browser.

A PDF version of this document is available on the Liquid Data documentation Home page on the e-docs Web site (and also on the documentation CD). You can open the PDF using Adobe Acrobat Reader and print the entire document (or a portion of it) in book format. To access the PDF files, open the Liquid Data documentation Home page, click PDF files and select the document you want to print.

If you do not have the Adobe Acrobat Reader, you can obtain a free version from the Adobe Web site at www.adobe.com.

Related Information

For more information about XQuery and XML Query languages, see the World Wide Web Consortium (W3C) Web site at http://www.w3.org/.

Contact Us!

Your feedback on the BEA Liquid Data documentation is important to us. Send us e-mail at **docsupport@bea.com** if you have questions or comments. Your comments will be reviewed directly by the BEA professionals who create and update the Liquid Data documentation.

In your e-mail message, please indicate that you are using the documentation for the BEA Liquid Data for WebLogic 1.0 release.

If you have any questions about this version of Liquid Data, or if you have problems installing and running Liquid Data, contact BEA Customer Support through BEA WebSupport at **www.bea.com**. You can also contact Customer Support by using the contact information provided on the Customer Support Card, which is included in the product package.

When contacting Customer Support, be prepared to provide the following information:

- Your name, e-mail address, phone number, and fax number
- Your company name and company address

- Your machine type and authorization codes
- The name and version of the product you are using
- A description of the problem and the content of pertinent error messages

Documentation Conventions

The following documentation conventions are used throughout this document.

Convention	Item
boldface text	Indicates terms defined in the glossary.
Ctrl+Tab	Indicates that you must press two or more keys simultaneously.
italics	Indicates emphasis or book titles.

`monospace text`	Indicates code samples, commands and their options, data structures and their members, data types, directories, and file names and their extensions. Monospace text also indicates text that you must enter from the keyboard.
	Examples:
	`#include <iostream.h> void main ( ) the pointer psz`
	`chmod u+w *`
	`\tux\data\ap`
	`.doc`
	`tux.doc`
	`BITMAP`
	`float`

`monospace boldface text`	Identifies significant words in code. *Example*: `void` **`commit`** `( )`
`monospace italic text`	Identifies variables in code. *Example*: `String` *`expr`*

Convention	Item
UPPERCASE TEXT	Indicates device names, environment variables, and logical operators. *Example*s: LPT1 SIGNON OR
`{ }`	Indicates a set of choices in a syntax line. The braces themselves should never be typed.
`[ ]`	Indicates optional items in a syntax line. The brackets themselves should never be typed. *Example*: `buildobjclient [-v] [-o name ] [-f file-list]... [-l file-list]...`
|	Separates mutually exclusive choices in a syntax line. The symbol itself should never be typed.
`...`	Indicates one of the following in a command line: • That an argument can be repeated several times in a command line • That the statement omits additional optional arguments • That you can enter additional parameters, values, or other information The ellipsis itself should never be typed. *Example*: `buildobjclient [-v] [-o name ] [-f file-list]... [-l file-list]...`
. . .	Indicates the omission of items from a code example or from a syntax line. The vertical ellipsis itself should never be typed.

CHAPTER 1

Understanding the Avitek Customer Self-Service Sample Application

This chapter provides a brief overview of the Avitek Customer Self-Service Sample Application (SampleApp), including a description of data sources, queries, and other application components necessary to develop the SampleApp.

About Avitek

Avitek is a retailer that has grown through acquisitions. Because of this growth it has two different order management systems (OMS) to manage electronics and apparel orders. These systems are under separate platforms and are not integrate with each other. Avitek has a customer relationship management (CRM) system to manage customer profile information. Finally, Avitek also has a Customer Service system to manage the support cases for Electronic products.

Avitek now want to build a customer self-service web site and this chapter will walk you through the the challenges in creating this application and how SampleApp was built to satisfy Avitek's needs.

Quick Start

To run the Avitek Customer Self-Service Sample Application:

1. Make sure you have installed Liquid Data 8.1 with samples (the full installation) into a WebLogic Platform 8.1 directory.
2. Start the Liquid Data Samples server, as described in Set Up and Run the Samples.
3. Run the SampleApp:

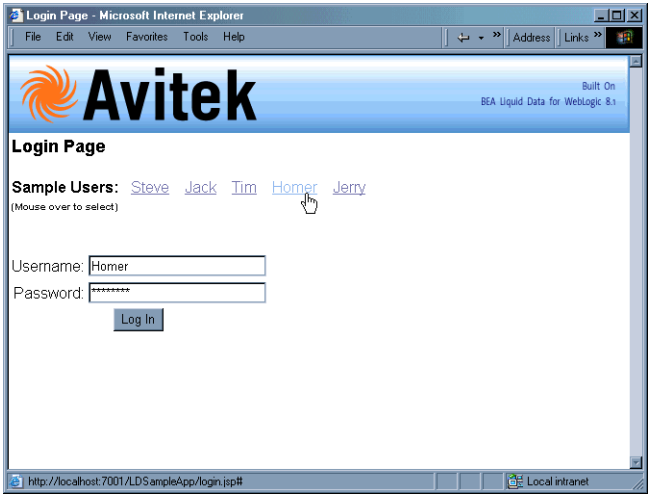
Start—>Programs—>BEA WebLogic Platform 8.1—>BEA Liquid Data for WebLogic 8.1—>Liquid Data Samples —>Sample Application Home

Alternatively, use the following URL to open the SampleApp login page:

```
http://localhost:7001/LDSampleApp/login.jsp
```

Log in using one of the names shown in the dialog box. Simply move your mouse over the login name of your choice to fill-in the name and password.

Figure 1-1 Sample App Login Page



Introduction

The Avitek Customer Self-Service Sample Application shows how you can use Liquid Data to solve the type of data integration problems frequently faced 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.
- Once you have such a single point of access to your data, can you develop reusable queries that are easily tested, stored, and retrieved?
- Once you have your query set, can you easily incorporate results into a widely available application?

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?

Case for an Avitek Self-Service Web Site

A survey commissioned by Marketing found Avitek 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 with Avitek customers. 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 …

Design Requirements

Site requirements seem simple. Customers need to:

- **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.

A business analyst develops the requirements for the application:

- A7/24 web site
- Less than 10 pages

- Low-maintenance
- Easily modified.

An application/UI designer starts “spec out" the required JSPs.

Information Technology (IT) Comes On Board: The Moment of Truth

Then an IT data architect analyzes the data requirements. This 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. 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.

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, but which also make yesterday’s data integration solution 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 SampleApp 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.

A Solution

When Avitek looked at Liquid Data, they found a product that addressed the underlying challenges posed by the apparently simple SampleApp:

- In Liquid Data, queries are described in simple, declarative text.
- Addressing the problem of data access, Liquid Data provides data integration through a highly-accessible graphical interface.
- Once data integration is achieved, persistent queries are generated.

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

Data Access. First, Liquid Data puts a common face on the data. This allows you 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 Liquid Data server.

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

The declarative form of Liquid Data artifacts (queries) makes them very readable. These queries are easily developed in the Data View Builder.

Query Deployment. Once developed, queries are easily integrated into a client application such as WebLogic Workshop.

Queries are as readily available to processes client applications thorough a variety of access methods such as an EJB API or a JSP tag library. Alternatively, these queries can also be accessed as web services.

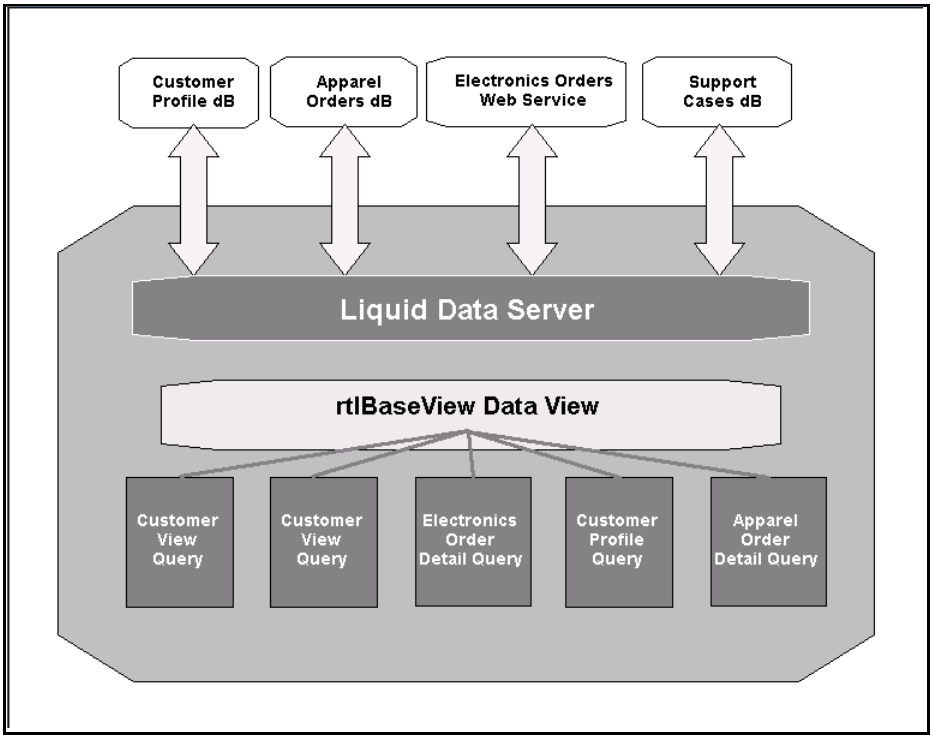
Integration. Business logic for the SampleApp is provided by the NetUI and page flow features of WebLogic Workshop.

However, Liquid Data queries are easily integrated and maintained within the business logic of any J2EE application.

Anatomy of the Avitek SampleApp

The SampleApp shows how Liquid Data-generated queries can aggregate data from potentially highly disparate data sources, allowing access to that data through a single point of access that itself is easily integrated with the application.

Figure 1-2 Avitek Queries Supported by the rtlBaseView



Note: To simplify the running of the SampleApp, the multiple data sources described in this document are simulated using the PointBase RDBMS which is shipped with Liquid Data. In the original implementation, these databases were represented by major vendor RDBMS systems.

Finding the Components

The SampleApp is located in the following directory:

```
<WL_HOME>/samples/liquiddata/SampleApp
```

SampleApp project files are available from:

```
<WL_HOME>/samples/liquiddata/SampleApp/DVBProject
```

SampleApp controls, pages, processes, and resources used to create the SampleApp can be found at:

```
<WL_HOME>/samples/liquiddata/SampleApp/LiquidDataSampleApp
```

Schemas used in the SampleApp are in the following directory:

```
<WL_HOME>/samples/domains/liquiddata/ldrepository/schemas/rtl
```

The Workshop `.work` file for the SampleApp is located at:

```
<WL_HOME>/samples/liquiddata/SampleApp/SampleApp.work
```

When you install Liquid Data with the WebLogic Platform, the source code for the SampleApp can be accessed from WebLogic Workshop. For instructions, see "Viewing the Avitek Customer Self-Service Sample Application Source" on page 1-16.

For additional information and references, see:

- "Running the Avitek Customer Self-Service Sample Application in a Browser" on page 1-14.
- "Using Workshop Controls to Develop Liquid Data Applications" in the *Application Developer's Guide* for the specific steps required to create the Liquid Data control.

Analyzing the SampleApp Architecture

Several BEA technologies are used in the Avitek Customer Self-Service Sample Application.

- **Query Development.** Queries and a data view that draw data from multiple data sources are created in the Data View Builder.
- **Query Access.** The WebLogic Workshop Liquid Data Control provides programmable access to Liquid Data queries.
- **Client-side Development**. WebLogic Workshop also provides an environment for building application logic through NetUI, page flow, and other technologies.

Steps Used in Developing the SampleApp

The basic steps used to develop the Avitek Customer Self-Service Sample Application were:

1. In Liquid Data:
   - Identify and analyze data sources
   - Configure Liquid Data to access these sources

2. In the Data View Builder:
   - Design and test queries and a data view
   - Deploy queries to the Liquid Data repository
3. In WebLogic Workshop:
   - Create a Liquid Data control
   - Add deployed queries to the Liquid Data control
   - Develop necessary client-side logic for the application
4. Test and deploy your application.

Available Data Sources

Although the SampleApp is very simple, the underlying data acquisition is potentially complex because date 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 `RTL-CUSTOMER`.
- **Order Management System (OMS).** Avitek has two order management systems:
  - Electronic products. OMS information is available from a legacy system via a web service. The web service has a method called `getOpenOrders()`, which takes a customer ID as input and returns a list of customer open order information through a web service, `ElectOrderService`.
  - Apparel products. Information is maintained on site in a second database. This is represented in Liquid Data as `RTL-APPL-OMS`.
- **Customer Service.** Service data is stored in a third database. The schema for this data is `RTL-SERVICE`.

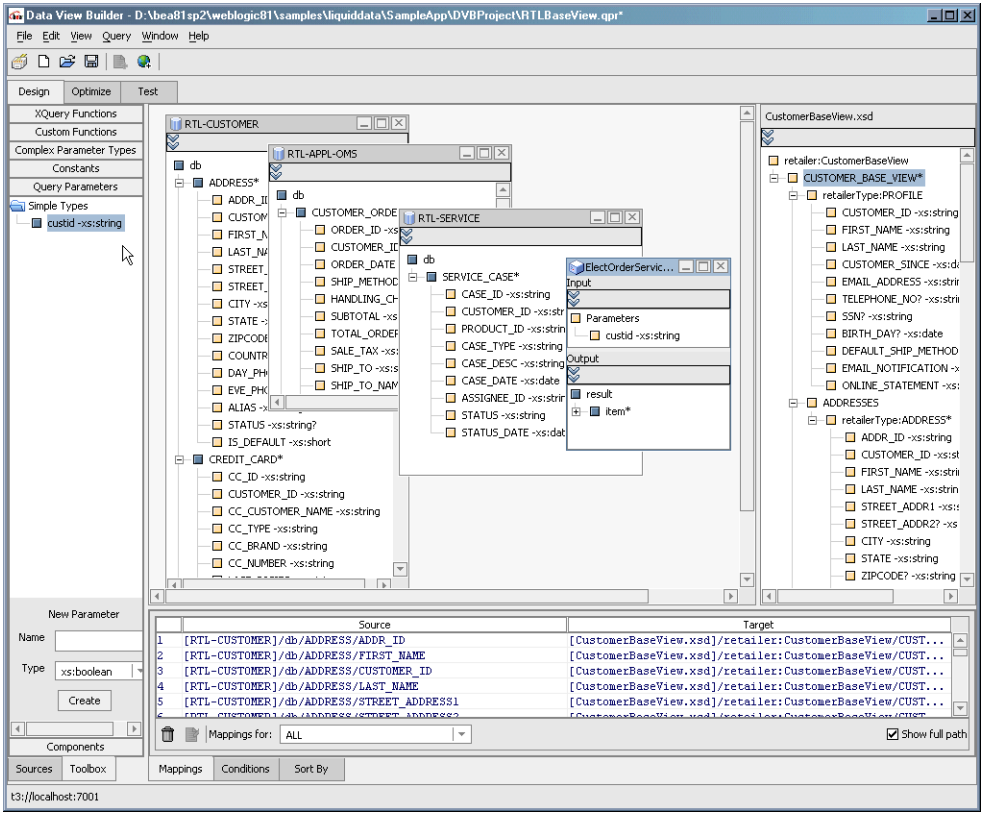
Avitek Customer Self-Service Sample Application Queries

The following section describes work done by the main SampleApp queries.

Data View Source (RTLBaseView.xq)

The RTLBaseView query forms the basis of the parameterized data view of the same name (`RTLBaseView.xv`) that supplies data for the Avetek SampleApp queries. Every element in the four underlying data sources is mapped to the `customerBaseView.xsd` target schema. A `custid` query parameter identifies the particular customer.

Figure 1-3 Project Showing RTLBaseView Query, the Basis for the Data View Underlying RTLSample Queries



A Data View Builder project is available that can help in understanding how this query was developed. You can review and run this project from:

```
<WL_HOME>/samples/liquiddata/SampleApp/DVBProject/RTLBaseView.qpr
```

Once perfected, the RTLBaseView.xq query is turned into a data view. Figure 1-4 shows the data view consolidates disparate data sources into a single complex object which can then be easily mapped to the target schema.

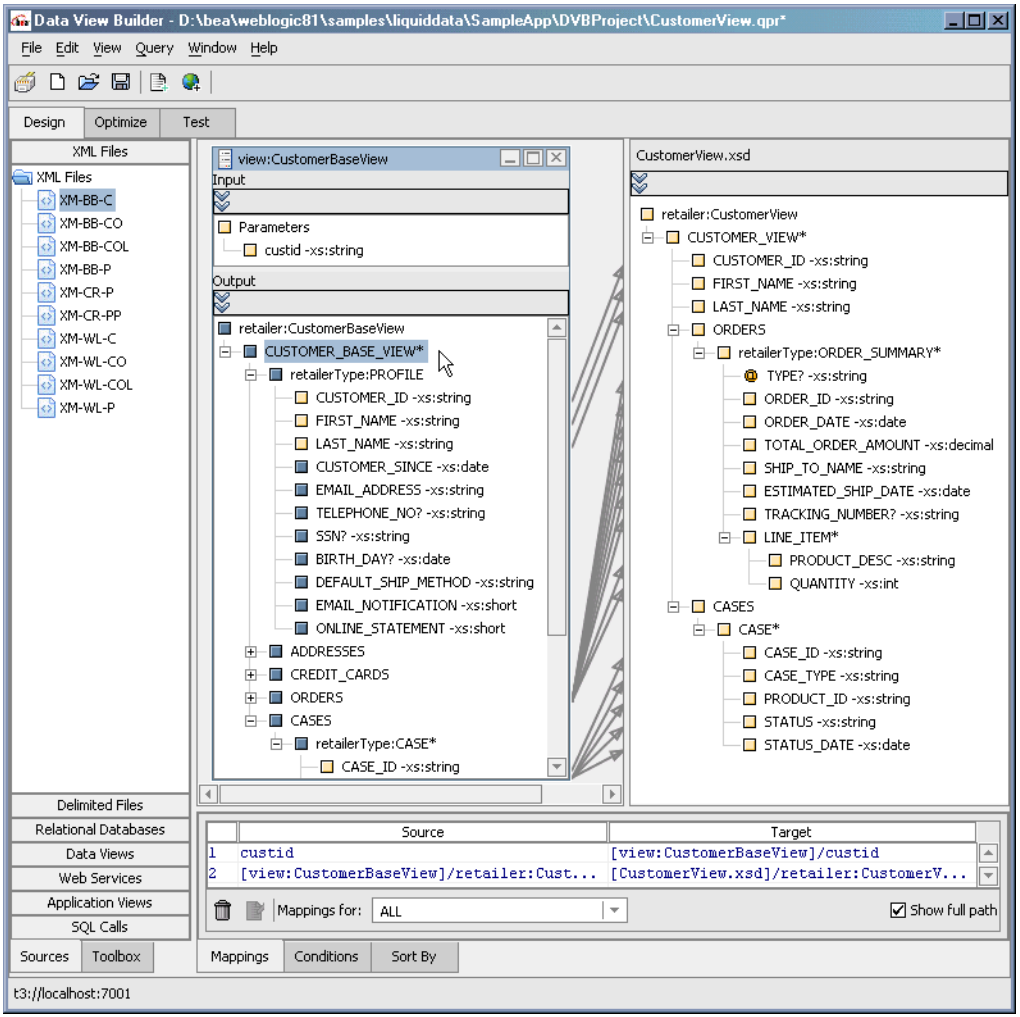
The CustomerView project (`CustomerView.qpr`) illustrates how the RTLBaseView data view is used to free the query developer from a need to directly access and map multiple data sources to the target schema. Instead, the RTLBaseView data source becomes a source schema that contains a superset of the data elements needed by any particular query.

Of course other data sources could also be used in creating a query but in this particular case a more effective design consolidating all the data elements into a since data view data source.

Default View (CustomerView.xq)

This query populates the home page of the SampleApp.

Figure 1-4 CustomerView Project Illustrating How the RTLBaseView Data View Supports Query Development



Using the RTLBaseView data view, the query returns:

- For a specific customer ID: FIRST_NAME and LAST_NAME.
- For Apparel open orders: ORDER_ID, ORDER_DATE, TOTAL_ORDER_AMOUNT, shipping information, and PRODUCT_DESC and QUANTITY for each items in the order.
- For Electronics open orders: the same information as is returned for open orders for apparel.
- For any customer support cases associated with the customer ID: CASE_ID, CASE_TYPE, PRODUCT_ID, STATUS, and STATUS_DATE.

A Data View Builder project is available that can help in understanding how this query was developed. You can review and run this project from:

```
<WL_HOME>/samples/liquiddata/SampleApp/DVBProject/CustomerView.qpr
```

Customer Profile (ProfileView.xq)

Retrieves information about customers from a RDBMS system maintained by the CRM software.

The query `ProfileView.xq` returns:

- For a given customer ID: name, email address, SSN, BIRTH_DATE, DEFAULT_SHIP_METHOD, and two Booleans identifying if the customer wants email notification of shipment and an online statement.
- For a given customer ID: address information such as ADDR_ID, CUSTOMER_ID, name, address, phone, ALIAS, STATUS, and whether the address is the default.
- For a given customer ID: credit card information such as CC_ID, CUSTOMER_NAME, CC_TYPE, CC_BRAND, LAST_DIGITS, EXPIRATION_DATE, ALIAS, STATUS, and ADDR_ID.

A Data View Builder project is available that can help in understanding how this query was developed. You can review and run this project from:

```
<WL_HOME>/samples/liquiddata/SampleApp/DVBProject/ProfileView.qpr
```

Electronic Division Order Detail (RTLElecOrderDetail.xq)

Provides customers with detailed information on any particular order selected from the History or Home page. Data originates in a proprietary order management system (OMS) which is made available to Liquid Data through a web service.

The query `ElecOrderDetail.xq` returns:

- For a given order ID: order information (ORDER_DATE, STATUS, and so forth).

- For a given customer ID and order ID: ship to information (CUSTOMER_ID, address information, STATUS, and so forth).
- For a given customer ID and order ID:
  - Bill to information (name of purchaser, DAY_PHONE, ALIAS, STATUS, IS_DEFAULT, and so forth).
  - Tracking information (TRACKING_NUMBER and information about each item in the order such as LINE_ITEM_ID, ORDER_ID, PRODUCT_ID, PRODUCT_DESC, QUANTITY, PRICE, and STATUS).

A Data View Builder project is available that you can use to understand how this query was developed. You can review and run this project from:

```
<WL_HOME>/samples/liquiddata/SampleApp/DVBProject/ElecOrderDetailView.qpr
```

Apparel Division Order Detail (AppOrderDetailView.xq)

Provides a customer with detailed information on any particular order found in the order history or on the Home page. The data originates in a RDBMS system.

The query `AppOrderDetailView.xq` returns the same data elements as `ElecOrderDetailView.xq`

A Data View Builder project is available for this query. You can review and run this project from:

```
<WL_HOME>/samples/liquiddata/SampleApp/DVBProject/ApplOrderDetailView.qpr
```

Order Summary (OrderSummaryView.xq)

Provides customers with summary information about all their orders with the company.

The query `RTLOrderSummaryView.xq` returns:

- For a range of dates: Apparel sale summary order information (ORDER_ID, ORDER_DATE, TOTAL_ORDER_AMOUNT, SHIP_TO_NAME, ESTIMATED_SHIP_DATE, TRACKING_NUMBER plus PRODUCT_DESC and QUANTITY for each items in the order).
- For a range of dates: Electronics sale summary information in the same form as for an Apparel sale (ORDER_ID, ORDER_DATE, and so forth).

A Data View Builder project is available for this query.You can review and run this project from:

```
<WL_HOME>/samples/liquiddata/SampleApp/DVBProject/OrderSummaryView.qpr
```

Table 1-5 shows the relationship between the data sources, major data elements (for RDBMS systems this means tables and columns), sample application JSPs, and the SampleApp web pages.

Table 1-5 SampleApp Primary Pages and Their Data Sources, Primary Data Elements, and Associated JSP

Customer Page	Data Source(s)	Primary Data Elements	JSP
Home page	• Customer Relationship Management (CRM) RDBMS • Order Management System (OMS) RDBMS • OMS via a web service • Customer Service RDBMS	from RTL_Customer • CustID, name from RTL_APPL_OMS • Order summary information from ElectOrdeService • Order summary information from RTL_SERVICE • Open case information	`DefaultView.jsp`
Profile page	Customer Relationship Management (CRM) RDBMS	from RTL_Customer • Customer profile • Bill-to, ship-to address • Credit card information	`ProfileView.jsp`
Details (accessible from Home page or History page)	**For apparel order detail:** • Order Management System (OMS) RDBMS • Customer Relationship Management (CRM) RDBMS **For electronics order detail:** • OMS via a web service • Customer Relationship Management (CRM) RDBMS	**For apparel order detail:** from RTL_APPL_OMS • Order detail information • tracking information from RTL_Customer • CustID, name, ship to address, billing information including last 5 digits of credit card **For electronics order detail:** (same as apparel order detail)	`OrderDetail.jsp`
History page	Same as Home page but no use of CS system	Same as Home page but shows no case information	`OrderHistory.jsp`

Running the Avitek Customer Self-Service Sample Application in a Browser

If you have not already done so, use "Quick Start" on page 1-2 to start the Avitek Customer Self-Service Sample Application. The following table shows the combination of login names and passwords:

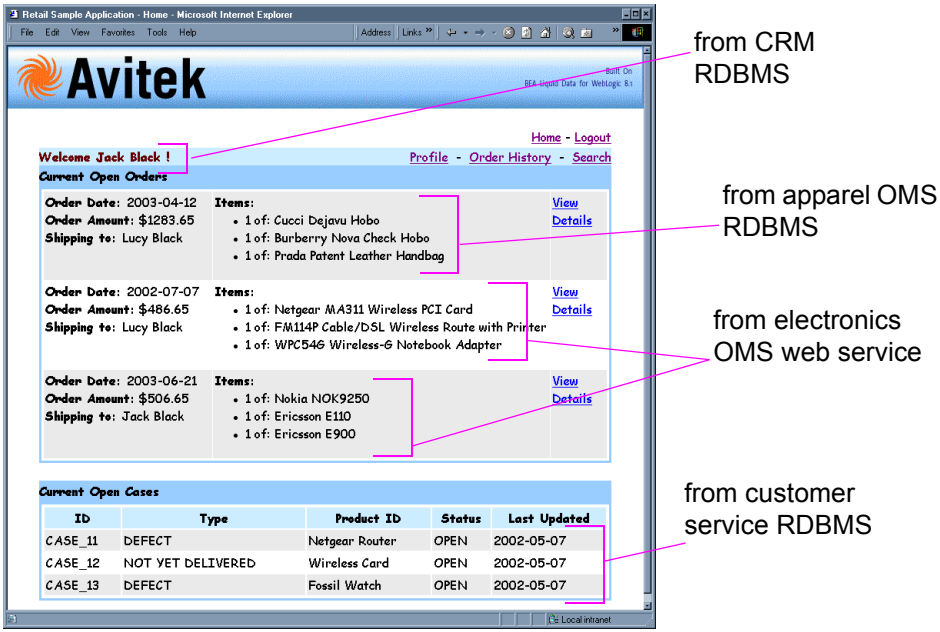
Table 1-6 Sample App Login Names and Passwords

Name	Password
Steve	steve123
Jack	jack1234
Tim	tim12345
Homer	homer123
Jerry	jerry123

Once a customer logs in, she or he sees the Customer Order Status screen (Figure 1-7). This page also serves as the Home page of the application.

SampleApp Home Page

Figure 1-7 SampleApp Home Page

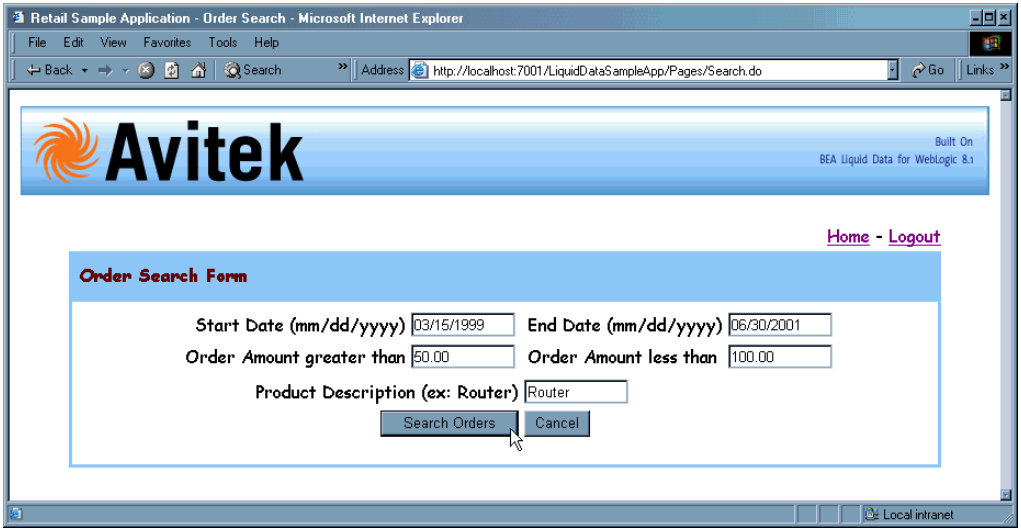


The Home page summarizes the customers order status. This customer service home page is controlled by a JSP called default.jsp. Call-outs show the underlying data sources. The page derives its font and other look-and-feel characteristics from a cascading stylesheet. For additional information on how this page is created, see "Page from the SampleApp" on page 1-28.

Order Search

In addition to being able to get Order History and Profile information from the SampleApp Home Page, the customer can search for specific orders based on order dates, items, or amounts.

Figure 1-8 Avitek Customer Self-Service Sample Application Search Form



The JSP for this page initiates a small Java program that incorporates the customers input and generates an XQuery based on the selected parameters. See "Ad Hoc Query" on page 1-28.

Viewing the Avitek Customer Self-Service Sample Application Source

The JSP pages and connection logic for the SampleApp were created in WebLogic Workshop. You can view the source for the SampleApp in WebLogic Workshop.

To open the SampleApp project, open the `SampleApp.work` file in WebLogic Workshop. The full path is:

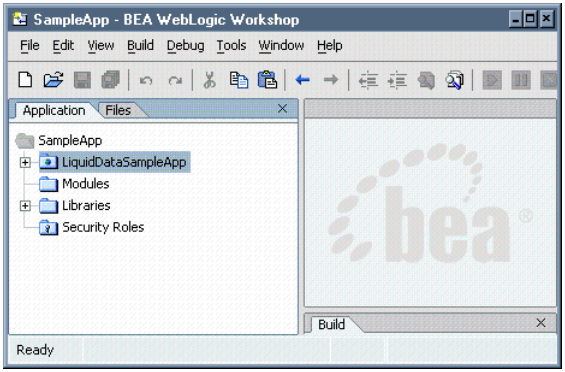
```
<WL_HOME>/samples/liquiddata/SampleApp/SampleApp.work
```

Note: If you are already running WebLogic Workshop or have previously run it with a different server setting, you may need to fix your application properties settings. If necessary, select the following from the WebLogic Workshop menu:

Tools —> Application Properties —> WebLogic Server

When opening the Avitek Customer Self-Service Sample Application in WebLogic Workshop, you initially see the application components and the work area.

Figure 1-9 Initial Avitek Customer Self-Service Sample Application Initial Project View



WebLogic Workshop allows you to work with the application you have under development as components or files. Web pages can be displayed and run, page flows and application logic can be developed and tested.

Note: Getting Started with WebLogic Workshop fully describes the development environment and how it can be used to build enterprise applications on the WebLogic Platform 8.1. The on-line document includes numerous examples and tutorials.

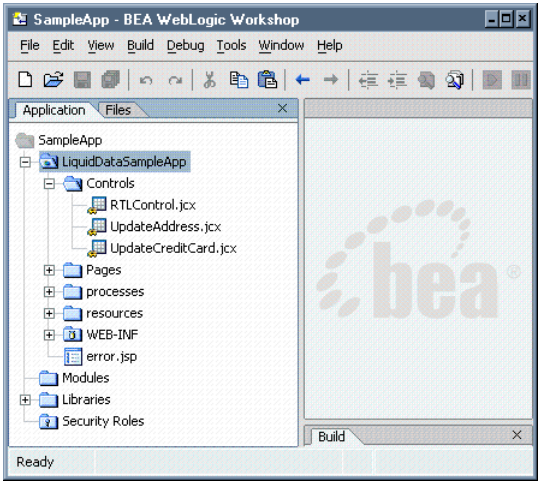
The WebLogic Workshop Samples application contains a number of samples relevant to data access and XML. See:

```
<WL_HOME>/samples/workshop/SamplesApp/SamplesApp.work
```

WebLogic Workshop Components of the SampleApp

All the components of the Avitek Customer Self-Service Sample Application are available from the Liquid Data Samples directory. See "Finding the Components" on page 1-6.

Figure 1-10 Avitek Customer Self-Service Sample Application Application Palette



When you click on the LiquidDataSampleApp folder in WebLogic Workshop, a number of components become visible. The following components are of special interest.

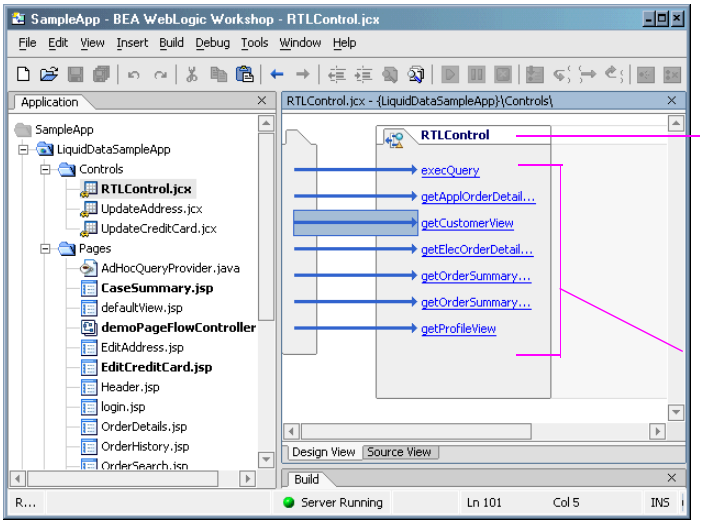
- **Controls folder.** For the SampleApp there are three controls:
  - `RTL_Control.jcx` This control contains methods for each stored query used in the Liquid Data application.
  - `UpdateAddress.jcx` This control contains methods that allow for updates to customer profile information.
  - `UpdateCreditCard.jcx` This is a control that contains methods to update customer credit card information.
- **Pages folder.** Contains the JSPs and the page flow controller, `demoPageFlowController.jpf`, that make up the SampleApp.
- **Resources folder.** Contains common resources such as the HTML style sheet, graphics files and common JSPs such as headers and footers.
- **WEB-INF folder.** Contains application source components, in particular struts and NetUI components that come with WebLogic Workshop.

Liquid Data Control

The RTLControl contains automatically-generated methods based on a set of Liquid Data stored queries selected by the developer of the control. For information on accessing Liquid Data queries in

WebLogic Workshop see “Select Queries to Add to a Control” in Using Workshop Controls to Develop Liquid Data Applications in the *Application Developer’s Guide*.

Figure 1-11 Design View of Liquid Data Control and Control Methods



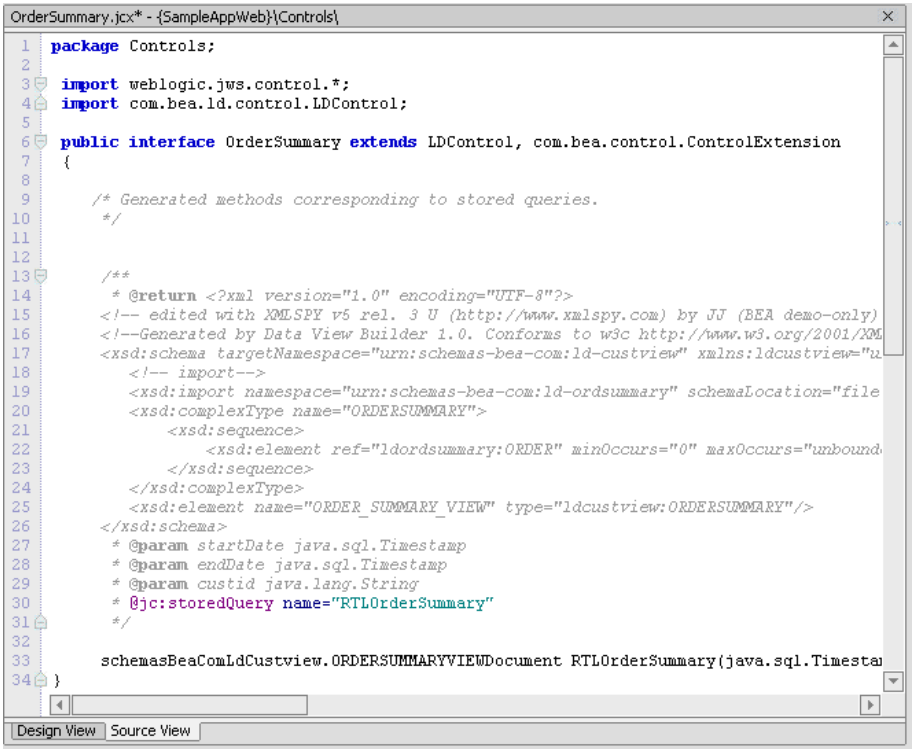
The method used by the client application to issue ad hoc queries against the Liquid Data server.

Methods based on deployed Liquid Data queries that are used by the application.

For information on deploying a Liquid Data stored query see Deploying a Query in the Testing Queries chapter of *Building Queries and Data Views*.

A portion of the Source View of the `RTLOrderSummary` method of the `RTLControl.jcx` file is shown in Figure 1-12. Note that the name of the target schema for the query appears in comments.

Figure 1-12 Source view of a Liquid Data Control

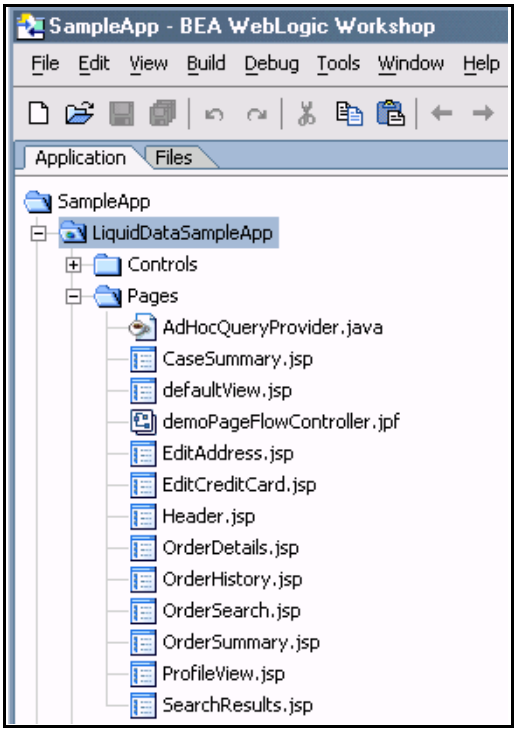


Each method in the Liquid Data Control corresponds to a stored query. Each method returns an `XMLBean` type. The XMLBeans are generated when the control is created, and are stored in the Libraries directory of the WebLogic Workshop application.

Application Pages

In the Pages folder you can find the Java Server Pages (JSPs) that make up the Avitek Customer Self-Service Sample Application. These were constructed entirely in WebLogic Workshop using queries from the Liquid Data control and NetUI graphical elements.

Figure 1-13 JSP Pages in the SampleApp



Application Logic: Page Flow

The Avitek Customer Self-Service Sample Application is composed of java server pages (JSPs) that are managed by a WebLogic Workshop PageFlowController. In the Avitek Customer Self-Service Sample Application the PageFlowController is named `demoPageFlowController.jpf`.

From an application logic perspective, whenever a user releases control of a page by selecting an option such as Next, Previous, Ok, Cancel, and so forth, application logic returns to the PageFlowController. Once that logic is processed, the user see with the appropriate web page.

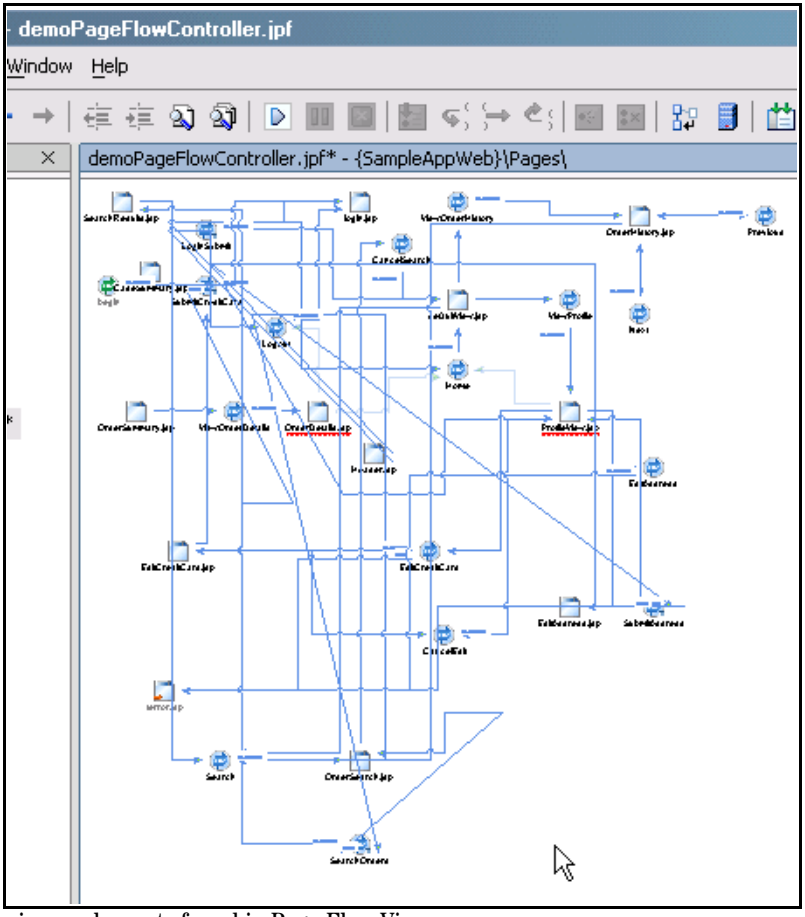
Using WebLogic Workshop you can inspect, change and extend page flow programmatically or graphically. There are three views of page flow: page flow, action, and source.

Page Flow: Flow View

The WebLogic Workshop page flow schematic (Figure 1-14) illustrates graphically the relationship between JSPs, including has modeless page flow is determined by user actions.

When you click on a particular page name, the page opens in the WebLogic Workshop development browser.

Figure 1-14 Avitek Customer Self-Service Sample Application Page Flow



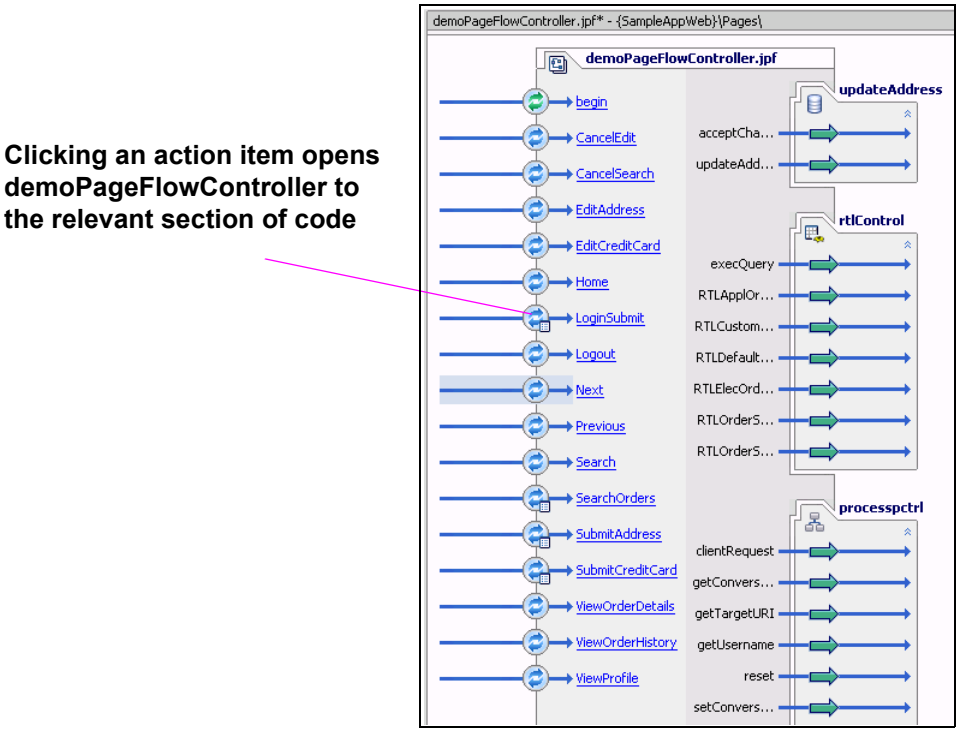
The three primary elements found in Page Flow View are:

- **Pages.** Clicking on a particular page will open the page for editing.
- **Actions.** The transition between pages is controlled by actions. Specific actions are user-driven such as in the case of a user accepting an option (“OK") or deciding against it (“Cancel") via a particular form or dialog box.
- **Source.** When source is selected, the code for the page appears in the work area.

Page Flow: Action View

The Action View tab is help in finding the location of page flow actions associated with the Avitek Customer Self-Service Sample Application.

Figure 1-15 Avitek Customer Self-Service Sample Application Action View



Page Flow: Source View

The PageFlowController file contains several parts:

- Declarations of graphical elements in the application.
- Public transient simple and array variables that facilitate and persist application logic between pages.
- `@jpf:forward` calls within comments that associate user actions with appropriate target JSPs.
- Queries and associated logic.

Inter-page Navigation

The code in Listing 1-1 shows the application's entry point and page flow options, as determined by whether the customer's login efforts qualify as `success` or `invalidLogin`.

Listing 1-1 also shows how customer details are obtained from Liquid Data through a query. Some things to note in examining this section of code are:

- A single `if/else if` statement allows you to pull data from two independent data sources.
- XQuery makes it possible to access data through Java rather than having to enter or preformat SQL statements.

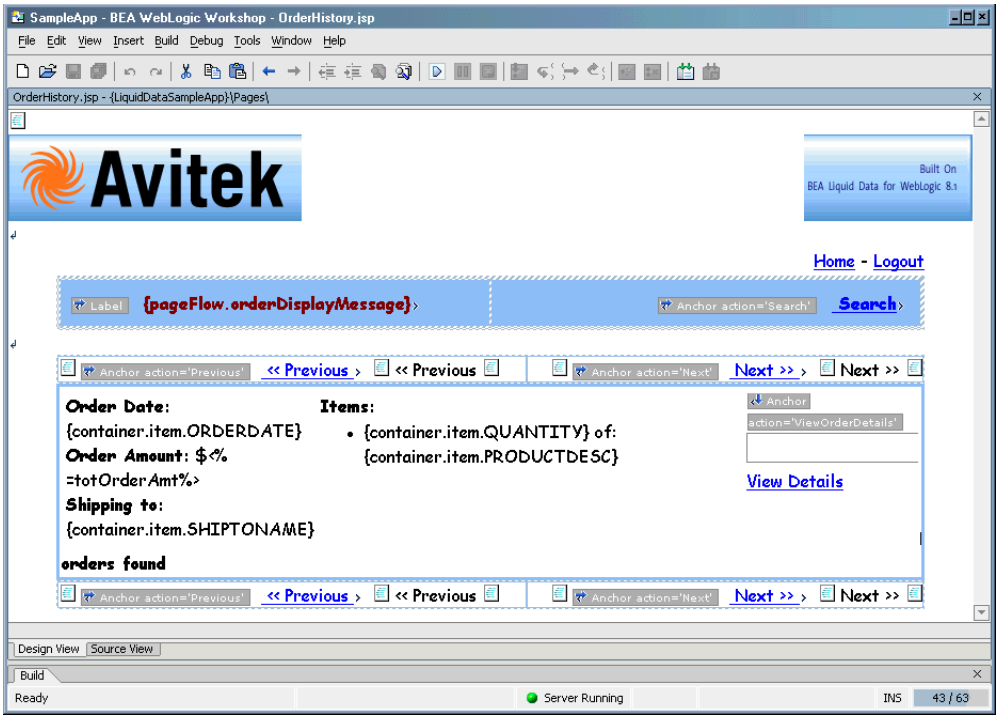
Listing 1-1 Page Flow Logic Supporting the OrderDetails.jsp

```
/**
     * @jpf:action
     * @jpf:forward name="success" path="OrderDetails.jsp"
     */
    protected Forward ViewOrderDetails()
    {
        String oId = getRequest().getParameter("orderId");
        String orderType = getRequest().getParameter("orderType");

        ORDERDETAILTYPE[] orders1 = null;

        if (orderType.equals("ELEC")){
            orders1 = rtlControl.getElecOrderDetailView(oId,
customer.getCUSTOMERID()).

getOrderDetailView().getORDERDETAILVIEWArray(0).getORDERDETAILArray();
        } else if (orderType.equals("APPL")){
            orders1 = rtlControl.getApplOrderDetailView(oId,
customer.getCUSTOMERID()).

getOrderDetailView().getORDERDETAILVIEWArray(0).getORDERDETAILArray();
        }
        order = (orders1 != null && orders1.length > 0) ? orders1[0] : null;
    ...
        return new Forward("success");
    }
```

Pagination

The OrderHistory page provides pagination; that is it retrieves data in batches and provides an interface to the user to get the next or previous batch of data.

Figure 1-16 Pagination, illustrated in OrderHistory.jsp.



When you click the Next or the Previous button a new parameterized query is executed that retrieves the next five or previous five records from both data sources, even though they do not have a common key between them. A local variable is incremented or decremented as part of the parameterization process.

Data Binding

In the Avitek Customer Self-Service Sample Application two types of data binding occur: read and update.

Read Cycle

When creating a page which reads data, you need to follow these steps:

1. Create the control to access the query you plan to use.
2. Create a page flow.
3. When the page is accessed, application logic invokes the query.
4. Data is bound to the JSP through NetUI.

Update Cycle

In the Avitek Customer Self-Service Sample Application update is accomplished in two ways.

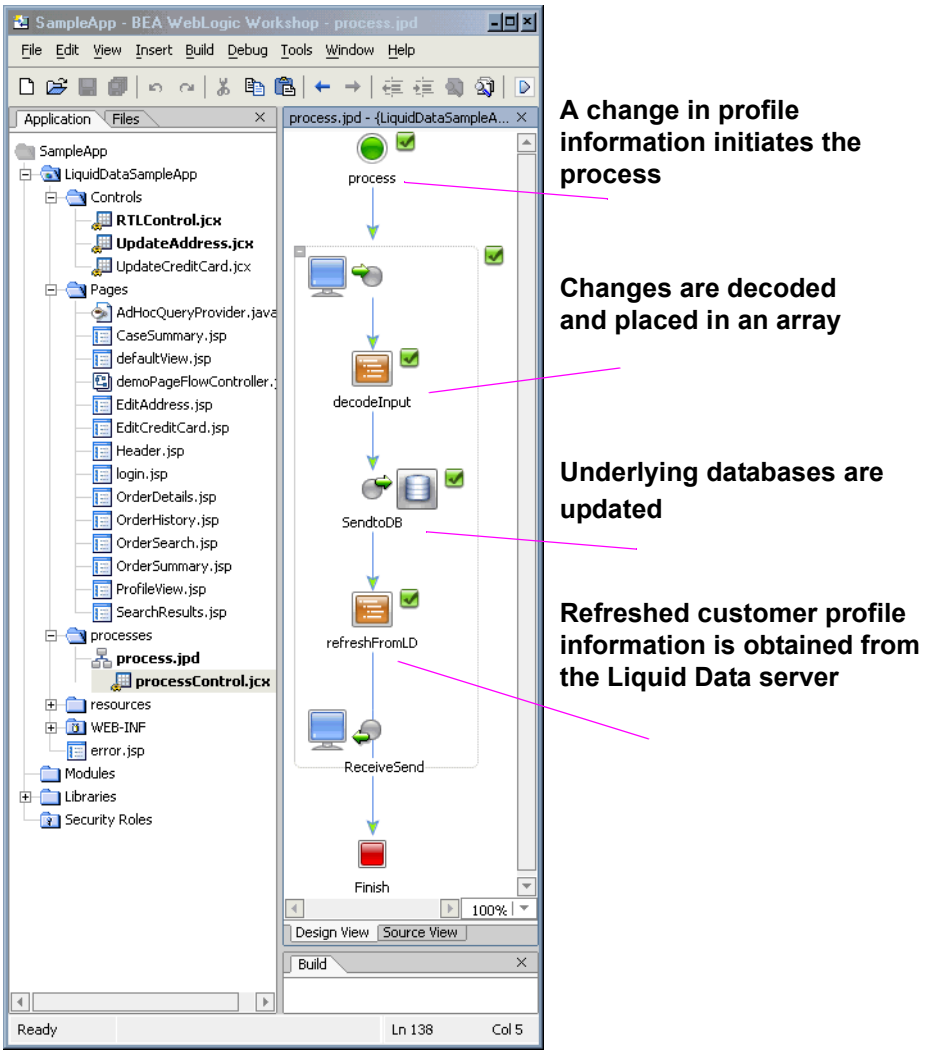
Updating Credit Card Information Using a WebLogic Workshop Database Control

When a customer changes his or her credit card information, a WebLogic Workshop database control.is used to update the data source using SQL. You can view source for this control in the file `process.jpd` in WebLogic Workshop processes folder.

Updating Customer Profile Information Using WebLogic Workshop Process Control

When the customer update profile information, the Liquid Data control invokes a WebLogic Integration (WLI) process that updates the source database and refreshes information in the application.

Figure 1-17 Design View of processControl.jcx



When activated, `processControl.jcx` completes its work by refreshing profile information through the Liquid Data server with this Liquid Data control call:

```
rtlControl.getProfileView(custId)
```

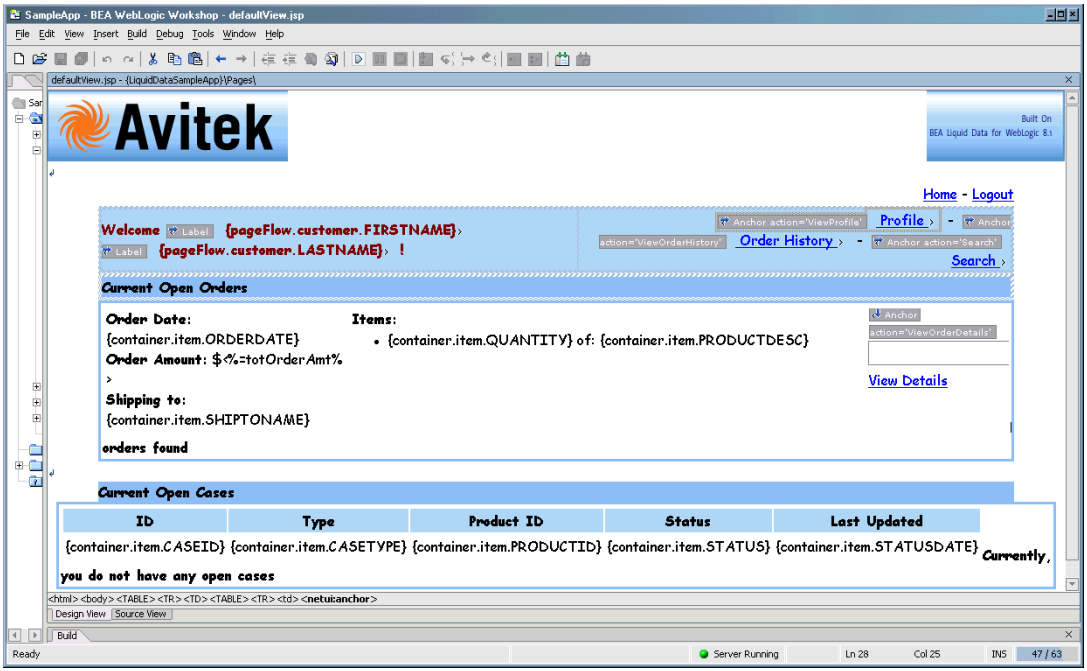
Ad Hoc Query

The `AdHocQueryProvider.java` has code to construct an XQuery based on inputs (such as an amount, date, or word) a user enters from a web page. Once created, the ad hoc query is called from `demoPageFlowController` as part of the `SearchOrders` routine.

Page from the SampleApp

The SampleApp Home page is called `default.jsp`

Figure 1-18 Design View of DefaultView.jsp, the SampleApp Home Page



Elements of `defaultView.jsp` were built through dragging-and-dropping variables of XMLBean type onto the JSP page. Other points of interest include:

- The NetUI repeater wizard was used to create the Current Open Order listing.
- The entire page is populated by a single Liquid Data data query, `getCustomerView.xq`, which takes a single `custID` parameter.

Summary

In summary, the Avitek Customer Self-Service Sample Application 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 Liquid Data queries and server for easy integration into applications or processes.

Where To Go From Here

Here are some additional resources for learning more about Liquid Data and WebLogic Workshop:

- "Using Liquid Data Controls to Develop Workshop Applications" in the *Application Developer's Guide* describes steps required to create the Liquid Data control.
- "Finding the Components" on page 1-6 lists where you can findSampleApp components and the Liquid Data project files that help in understanding how the underlying queries were developed.
- To learn more about WebLogic Workshop see Getting Started with WebLogic Workshop.
- To learn more about XML Beans see Getting Started with XML Beans.
- The WebLogic Workshop Samples application contains a number of samples relevant to data binding and XML. See:

  ```
  <WL_HOME>/samples/workshop/SamplesApp/Samples.work
  ```

CHAPTER 2

Query Cookbook

This section provides examples of BEA Liquid Data for WebLogic queries using some of the advanced features and tools offered in the Data View Builder. This book assumes that you are familiar with the Data View Builder user interface and that you have an understanding of the basic concepts and tasks using the Data View Builder. For details on using the Data View Builder, see *Building Queries and Data Views*.

The following use cases and examples are provided here to give you a jump-start for constructing real-world queries to solve common problems. Each use case includes a viewlet demo of building the solution using Data View Builder. Watching a viewlet takes 3 to 5 minutes.

- Example 1: Simple Joins (View a Demo)
- Example 3: Aggregates (View a Demo)
- Example 4: Date and Time Duration (View a Demo)
- Example 5: Union (View a Demo)
- Example 6: Minus (View a Demo)
- Example 7: Complex Parameter Type (CPT) (View a Demo)

For an example of using a stored procedure in a query, see "Example: Defining and Using a Customer Orders Stored Procedure."

Each use case has an example with a description of the problem and the steps to solve the problem. The examples use two databases:

- The BroadBand database (`PB-BB`) contains "BroadBand" subscribers and service orders

- The Wireless database (`PB-WL`) contains "Wireless" subscribers.

In cases where the target schemas do not already exist in the Liquid Data Samples Server repository, they are provided in this documentation along with the examples. You can cut-and-paste the schema content into a `.xsd` file to construct your own target schemas. (You can also copy from the PDF version of this document which may give you a copy that formats better your text editor.)

Note: To find out what data are contained in any data source, create a new "test" project, open the source schema you are interested in, and map key source nodes to any appropriate target schema. (For example, map customer first and last names and customer ID from source to target schemas.) Then click on Test tab and choose Query—>Run Query. The result will return all customers in the data source queried.

As you work through the examples, remember to save any projects that you want to keep before creating new ones.

Example 1: Simple Joins

A join merges data from two data sources based on a certain relation.

The Problem

For each Wireless Customer ID, determine whether the customer has any BroadBand orders. Assume that the Customer ID matches across databases.

The Solution

First, you want to find matching BroadBand customers (who are also included in the Wireless database), then return BroadBand Order IDs for the matching customers. Because Customer IDs in the Wireless database align with those in BroadBand, we can find matching BroadBand customers with a simple join of Wireless Customer IDs with the Customer IDs in the BroadBand order information.

To create the solution, follow these steps:

- View a Demo
- Ex 1: Step 1. Verify the Target Schema is Saved in Repository
- Ex 1: Step 2. Open Source and Target Schemas
- Ex 1: Step 3. Map Nodes from Source to Target Schema to Project the Output
- Ex 1: Step 4. Create a Query Parameter for a Customer ID to be Provided at Query Runtime

View a Demo

Simple Joins Demo... If you are looking at this documentation online, you can click the "Demo" button to see a viewlet demo showing how to build the conditions and create the mappings described in this example. This demo previews the steps described in detail in the following sections. The demo assumes you already have the target schema in the Liquid Data Samples Server repository.

Ex 1: Step 1. Verify the Target Schema is Saved in Repository

For this example, we will use a target schema called `customerOrders.xsd`. This schema is available in the Liquid Data Samples Server repository. The path to the schemas folder is:

```
<WL_HOME>/samples/domains/liquiddata/ldrepository/schemas/
```

Just in case you want to verify that you have the right schema file, the following code listing shows the XML for this schema.

Listing 2-1 XML Source for customerOrders.xsd Target Schema File

```
<?xml version = "1.0" encoding = "UTF-8"?>

<xsd:schema xmlns:xsd = "http://www.w3.org/2001/XMLSchema">
   <xsd:element name = "customers">
      <xsd:complexType>
         <xsd:sequence>
            <xsd:element ref = "customer" minOccurs = "0" maxOccurs =
"unbounded"/>
         </xsd:sequence>
      </xsd:complexType>
   </xsd:element>
   <xsd:element name = "customer">
      <xsd:complexType>
         <xsd:sequence>
            <xsd:element ref = "first_name"/>
            <xsd:element ref = "last_name"/>
            <xsd:element ref = "orders" minOccurs = "0" maxOccurs = "unbounded"/>
         </xsd:sequence>
```

```
         <xsd:attribute name = "id" use = "optional" type = "xsd:string"/>
      </xsd:complexType>
   </xsd:element>
   <xsd:element name = "first_name" type = "xsd:string"/>
   <xsd:element name = "last_name" type = "xsd:string"/>
   <xsd:element name = "orders">
      <xsd:complexType>
         <xsd:sequence>
            <xsd:element ref = "order" minOccurs = "0" maxOccurs = "unbounded"/>
         </xsd:sequence>
      </xsd:complexType>
   </xsd:element>
   <xsd:element name = "order">
      <xsd:complexType>
         <xsd:attribute name = "id" use = "optional" type = "xsd:string"/>
         <xsd:attribute name = "date" use = "optional" type = "xsd:string"/>
         <xsd:attribute name = "amount" use = "optional" type = "xsd:string"/>
      </xsd:complexType>
   </xsd:element>
</xsd:schema>
```

Ex 1: Step 2. Open Source and Target Schemas

1. In the Data View Builder, choose File—>New Project to clear your desktop and reset all default values.
2. On the Builder Design—>Sources tab, click Relational Databases and open two data sources:
   - Double-click on the `PB-WL` (Wireless) relational database to open the schema for this data source.
   - Double-click on the `PB-BB` (BroadBand) relational database to open the schema for this data source.
3. Choose the menu option File—>Set Target Schema.

   Navigate to the Liquid Data Samples Server repository or to the location where you saved the `customerOrders.xsd` schema. Choose `customerOrders.xsd` and click Open.

   `customerOrders.xsd` appears as the target schema.

   This target schema is displayed as a docked schema window on the right side of the design area.
4. Click the plus (+) sign (or right-mouse click and choose Expand) to expand the nodes in each source schema and in the target schema.

Ex 1: Step 3. Map Nodes from Source to Target Schema to Project the Output

1. Drag and drop `[PB-WL]db/CUSTOMER/CUSTOMER_ID` from source schema onto the target schema `[customerOrders.xsd]/customers/customer/id`
2. Drag and drop `[PB-WL]db/CUSTOMER/FIRST_NAME` from source schema onto the target schema `[customerOrders.xsd]/customers/customer/first_name`
3. Drag and drop `[PB-WL]db/CUSTOMER/LAST_NAME` from source schema onto the target schema `[customerOrders.xsd]/customers/customer/last_name`
4. Drag and drop `[PB-BB]db/CUSTOMER_ORDER/ORDER_DATE` onto the target schema `[customerOrders.xsd]customers/customer/orders/order/order_date`
5. Drag and drop `[PB-BB]db/CUSTOMER_ORDER/ORDER_ID` onto the target schema `[customerOrders.xsd]customers/customer/orders/order/id`

Ex 1: Step 4. Create a Query Parameter for a Customer ID to be Provided at Query Runtime

Create a Query Parameter wireless_id variable for a Wireless Customer ID that you will supply at query execution time:

1. On the Builder Design, click Toolbox and then click Query Parameter.
2. From the Type drop-down menu, choose `xs:string`.
3. In Parameter Name field, enter `wireless_id` and click Add.

   The new parameter is displayed in the Query Parameters tree.

Ex 1: Step 5. Assign the Query Parameter to a Source Node

Drag and drop the wireless_id query parameter to [PB-WL]db/CUSTOMER/CUSTOMER_ID.

Ex 1: Step 6. Join the Wireless and BroadBand Customer IDs

Drag and drop `[PB-WL]db/CUSTOMER/CUSTOMER_ID` to `[PB-BB]db/CUSTOMER_ORDER/CUSTOMER_ID`

Ex 1: Step 7. View the XQuery and Run the Query to Test it

1. Click on the Test tab.

   The generated XQuery for this query is shown in the following code listing.

Listing 2-2 XQuery for Example 1: Simple Joins

```
<customers>
   {
   for $PB_WL.CUSTOMER_1 in document("PB-WL")/db/CUSTOMER
   where ($#wireless_id of type xs:string  eq  $PB_WL.CUSTOMER_1/CUSTOMER_ID)
   return
   <customer id={$PB_WL.CUSTOMER_1/CUSTOMER_ID}>
      <first_name>{ xf:data($PB_WL.CUSTOMER_1/FIRST_NAME) }</first_name>
      <last_name>{ xf:data($PB_WL.CUSTOMER_1/LAST_NAME) }</last_name>
      <orders>
         {
         for $PB_BB.CUSTOMER_ORDER_2 in document("PB-BB")/db/CUSTOMER_ORDER
         where ($PB_WL.CUSTOMER_1/CUSTOMER_ID  eq
$PB_BB.CUSTOMER_ORDER_2/CUSTOMER_ID)
         return
         <order id={$PB_BB.CUSTOMER_ORDER_2/ORDER_ID} date={cast as
xs:string($PB_BB.CUSTOMER_ORDER_2/ORDER_DATE)}></order>
         }
      </orders>
   </customer>
   }
</customers>
```

2. Set the variable value to submit to the query when the query runs. To do this, you need to enter a value in the Query Parameter panel. Click into the cell under Value and enter CUSTOMER_3.

   (Customer IDs CUSTOMER_1 through CUSTOMER_10 are available in the data source to try.)

3. Click the Run query button to run the query against the data sources.

Ex. 1: Step 8. Verify the Result

Running this query with the `wireless_id` parameter set to CUSTOMER_3 produces the following XML query result.

Listing 2-3 Result for Example 1: Simple Joins

```
<customers>
  <customer id="CUSTOMER_3">
    <first_name>JOHN_3</first_name>
    <last_name>KAY_3</last_name>
```

```
    <orders>
      <order date="2002-03-06-08:00" id="ORDER_ID_3_0"/>
      <order date="2002-03-06-08:00" id="ORDER_ID_3_1"/>
      <order date="2002-03-06-08:00" id="ORDER_ID_3_2"/>
      <order date="2002-03-06-08:00" id="ORDER_ID_3_3"/>
    </orders>
  </customer>
</customers>
```

Example 2: Retrieving Information

The Problem

Find Customer IDs for customers who have both Wireless and BroadBand accounts and include in the generated data the state each customer resides in.

The Solution

This example shows how to do the following:

- Project output
- Specify the order of the result

Open Data Sources and Add a Target Schema

1. Choose File —> New Project to clear your desktop and reset all default values.
2. On the Builder Toolbar —> Sources tab, click Relational Databases and open two data sources:
   - Double-click on the `PB-WL` relational database to open the schema for this data source.
   - Double-click on the `PB-BB` relational database to open the schema for this data source.
3. Choose File —> Set Target Schema. Use the file browser to navigate to the Liquid Data Samples Server repository and select `amtByState.xsd` as the target schema.

   Note: If `amtByState.xsd` is not already available from the repository, you can create and save it yourself. For a copy of the schema file and instructions on how to save it to the Liquid Data Samples Server repository, see "Target Schemas" in *Building Queries and Data Views*. The schema itself is shown in Listing 5-1 of the same book.

   This target schema is displayed as a docked schema window on the right side of the workspace.

Map Elements from Source to Target Schema to Project Output

To project Customer first and last names and state to Target, do the following:

1. Drag and drop Wireless `[PB-WL]/FIRST_NAME` (under `CUSTOMER*`) onto `FIRST_NAME` in the Target schema.
2. Drag and drop Wireless `[PB-WL]/LAST_NAME` (under `CUSTOMER*`) onto `LAST_NAME` in the Target schema.

3. Drag and drop Wireless `[PB-WL]/STATE` (under `CUSTOMER*`) onto `STATE` (under `CUSTOMER*`) in the Target schema.

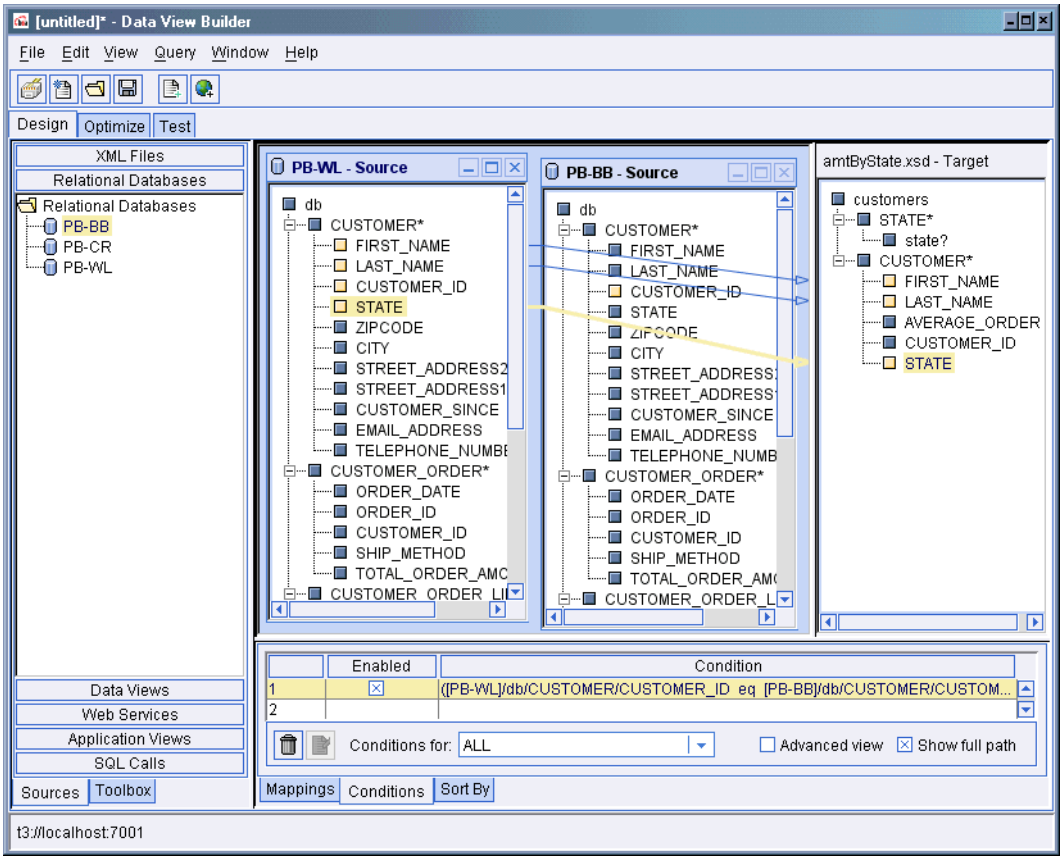
Join Two Sources

To create a *join* between Wireless `[PB-WL]` and BroadBand `[PB-BB]` on customer IDs, do the following:

- Drag and drop BroadBand `[PB-BB]/CUSTOMER_ID` (under CUSTOMER*) onto the associated Wireless `[PB-WL]/CUSTOMER_ID` element.

The following shows the mappings in the Data View Builder.

Figure 2-1 Query to Identify Customers by Customer ID and Sort by State



Specify the Order of the Result Using the Sort By Features

To order the output alphabetically by State do the following:

1. Click the Sort By tab.

   This tab shows repeatable elements in the target schema with subordinate fields that you can select for ordering.

2. From the drop-down menu choose CUSTOMER, and then click into the Direction cell next to STATE and set STATE to Ascending.

   This will cause the query to display the results in ascending order by state.

View and Run the Query

Now that you have built the query, you can switch to the Test tab to view the generated XQuery and run the query to see the kind of result it returns.

1. Click on the Test tab.

   The generated XQuery for this query is shown in the following code listing.

Listing 2-4 XQuery for Example: Query Customers by ID and Sort by State

```
<customers>
   {
   for $PB_WL.CUSTOMER_1 in document("PB-WL")/db/CUSTOMER
   where xf:not(xf:empty(
      for $PB_BB.CUSTOMER_2 in document("PB-BB")/db/CUSTOMER
      where ($PB_BB.CUSTOMER_2/CUSTOMER_ID  eq  $PB_WL.CUSTOMER_1/CUSTOMER_ID)
      return
      xf:true()))
   return
   <CUSTOMER>
      <FIRST_NAME>{ xf:data($PB_WL.CUSTOMER_1/FIRST_NAME) }</FIRST_NAME>
      <LAST_NAME>{ xf:data($PB_WL.CUSTOMER_1/LAST_NAME) }</LAST_NAME>
      <STATE>{ xf:data($PB_WL.CUSTOMER_1/STATE) }</STATE>
   </CUSTOMER>
   sortby(STATE ascending)
   }
</customers>
```

2. Click the Run query button to run the query against the data sources.

Querying these data sources as described in this example produces the XML query result shown in the following code listing.

Figure 2-2 XML Result for Example: Query Customers by ID and Sort by State

```
<customers>
  <CUSTOMER>
    <FIRST_NAME>JOHN_3</FIRST_NAME>
    <LAST_NAME>KAY_3</LAST_NAME>
    <STATE>AZ</STATE>
  </CUSTOMER>
  <CUSTOMER>
    <FIRST_NAME>JOHN_8</FIRST_NAME>
    <LAST_NAME>KAY_8</LAST_NAME>
    <STATE>AZ</STATE>
  </CUSTOMER>
  <CUSTOMER>
    <FIRST_NAME>JOHN_10</FIRST_NAME>
    <LAST_NAME>KAY_10</LAST_NAME>
    <STATE>CA</STATE>
  </CUSTOMER>
  <CUSTOMER>
    <FIRST_NAME>JOHN_5</FIRST_NAME>
    <LAST_NAME>KAY_5</LAST_NAME>
    <STATE>CA</STATE>
  </CUSTOMER>
  <CUSTOMER>
    <FIRST_NAME>JOHN_4</FIRST_NAME>
    <LAST_NAME>KAY_4</LAST_NAME>
    <STATE>NV</STATE>
  </CUSTOMER>
  <CUSTOMER>
    <FIRST_NAME>JOHN_9</FIRST_NAME>
    <LAST_NAME>KAY_9</LAST_NAME>
    <STATE>NV</STATE>
  </CUSTOMER>
  <CUSTOMER>
    <FIRST_NAME>JOHN_1</FIRST_NAME>
    <LAST_NAME>KAY_1</LAST_NAME>
    <STATE>TX</STATE>
  </CUSTOMER>
  <CUSTOMER>
    <FIRST_NAME>JOHN_6</FIRST_NAME>
    <LAST_NAME>KAY_6</LAST_NAME>
    <STATE>TX</STATE>
  </CUSTOMER>
  <CUSTOMER>
    <FIRST_NAME>JOHN_2</FIRST_NAME>
    <LAST_NAME>KAY_2</LAST_NAME>
    <STATE>WA</STATE>
  </CUSTOMER>
  <CUSTOMER>
    <FIRST_NAME>JOHN_7</FIRST_NAME>
```

```
    <LAST_NAME>KAY_7</LAST_NAME>
    <STATE>WA</STATE>
  </CUSTOMER>
</customers>
```

Example 3: Aggregates

Aggregate functions produce a single value from a set of input values. An example of an aggregate function in Data View Builder is the count function, which takes a list of values and returns the number of values in the list.

The Problem

Find the number of orders placed in the BroadBand database for a given customer who is also in the Wireless database.

The Solution

This query relies on a data view called AllOrders which retrieves customers who are in the BroadBand database and also in the Wireless database. For each of these customers, the customer ID and orders are retrieved. Then, we use the Aggregate function `count` to determine how many orders are associated with a given customer. At query runtime, a customer ID is submitted as a query parameter and the result returns the number of orders associated with the given customer ID.

To create the solution, follow these steps:

- View a Demo
- Ex 3: Step 1. Configure the “AllOrders” Stored Query as a Data View
- Ex 3: Step 2. Restart the Data View Builder and Find the New Data View
- Ex 3: Step 3. Verify that the Target Schema is Saved in the Repository
- Ex 3: Step 4. Open the Data Sources and Target Schema
- Ex 3: Step 5. Map Source Nodes to Target to Project the Output
- Ex 3: Step 6. Create Two Query Parameters to be Provided at Query Runtime
- Ex 3: Step 7. Assign the Query Parameters to Source Nodes
- Ex 3: Step 8. Add the Count XQuery Function
- Ex 3: Step 9. Verify Mappings and Conditions
- Ex 3: Step 10. View the XQuery and Test by Running the Query
- Ex 3: Step 11. Verify the Result

View a Demo

Aggregates Demo... If you are looking at this documentation online, you can click the "Demo" button to see a viewlet demo showing how to build the conditions and create the mappings described in this example. This demo previews the steps described in detail in the following sections. The demo assumes you already have the target schema in the Liquid Data Samples Server repository and have created and configured the data view data source required for this example.

Ex 3: Step 1. Configure the "AllOrders" Stored Query as a Data View

For this example, we will use a data view data source called `AllOrders.xv`. However, before this data view can be used, it must first be created.

Figure 2-3 XML Source for AllOrders.xv Data View File

```
<customers>
   {
   for $PB-BB.CUSTOMER_1 in document("PB-BB")/db/CUSTOMER
   for $PB-WL.CUSTOMER_2 in document("PB-WL")/db/CUSTOMER
   where ($PB-WL.CUSTOMER_2/CUSTOMER_ID  eq $PB-BB.CUSTOMER_1/CUSTOMER_ID)

   return
   <customer id={$PB-WL.CUSTOMER_2/CUSTOMER_ID}>
      <first_name>{ xf:data($PB-WL.CUSTOMER_2/FIRST_NAME) }</first_name>
      <last_name>{ xf:data($PB-WL.CUSTOMER_2/LAST_NAME) }</last_name>
      <orders>
         {
         for $PB-BB.CUSTOMER_ORDER_4 in document("PB-BB")/db/CUSTOMER_ORDER
         where ($PB-BB.CUSTOMER_1/CUSTOMER_ID  eq
$PB-BB.CUSTOMER_ORDER_4/CUSTOMER_ID)
         return
         <order id={$PB-BB.CUSTOMER_ORDER_4/ORDER_ID}
date={$PB-BB.CUSTOMER_ORDER_4/ORDER_DATE}></order>
         }
      </orders>
   </customer>
   }
</customers>
```

Use the WLS Administration Console to Configure Your Data View Data Source

1. Start and login to the WLS Administration Console for the Samples server you are using.

To start the WLS Administration Console for the Liquid Data Samples server running on your local machine, type the following URL in a browser address field:

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

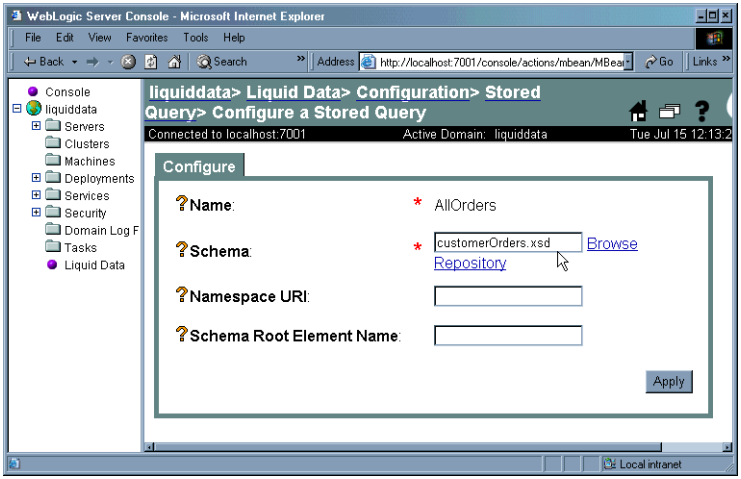
Login to the console by providing the following default username and password for the Samples server.

Table 2-4 User Name and Password for Samples WLS Administration Console

Field	Defaults
Username	`system`
Password	`security`

2. In the left pane, click the Liquid Data node.
3. In the right pane click the Stored Queries tab.
4. Locate `AllOrders` among the queries (it will be near the bottom of the listing).
5. Click the Configure link.
6. For a schema enter `customerOrders.xsd` or browse to the file. (You do not need to enter a namespace URI or schema root element name for this example.)
7. Click Create.

Figure 2-5 Configuring Liquid Data Source Description for a Data View



8. Find all orders in the list of stored queries (now it should be near the top of the listing).
9. Click the Create Data View link.

   The Create a Data View dialog box is displayed.
10. Provide a name such as `AllOrders` for your new data view, then click Create.

    Available data views will display. `AllOrders` should appear in the list.

Ex 3: Step 2. Restart the Data View Builder and Find the New Data View

1. Reconnect to the Data View Builder by selecting File—>Connect.
2. On the Design tab, on the Builder Toolbar, click the Sources tab, then click Data Views.

   The `AllOrders.xv` data view should be displayed in the list of available data views.

Ex 3: Step 3. Verify that the Target Schema is Saved in the Repository

For this example, we will use a target schema called `customerOrdersA.xsd`. This schema is available in the Liquid Data Samples Server repository. The path to the schemas folder is:

```
<WL_HOME>/samples/domains/liquiddata/ldrepository/schemas/
```

Just in case you want to verify that you have the right schema file, the following code listing shows the XML for this schema.

Listing 2-5 XML Source for customerOrdersA.xsd Target Schema File

```
<?xml version = "1.0" encoding = "UTF-8"?>

<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
   <xsd:element name="customers">
      <xsd:complexType>
         <xsd:sequence>
            <xsd:element name="customer" minOccurs="0" maxOccurs="unbounded">
               <xsd:complexType>
                  <xsd:sequence>
                     <xsd:element name="first_name" type="xsd:string"/>
                     <xsd:element name="last_name" type="xsd:string"/>
                     <xsd:element name="orders" minOccurs="0"
maxOccurs="unbounded">
                        <xsd:complexType>
                           <xsd:sequence>
                              <xsd:element name="order" minOccurs="0"
maxOccurs="unbounded">
                                 <xsd:complexType>
                                    <xsd:sequence>
                                    </xsd:sequence>
                                 <xsd:attribute name="id" type="xsd:string"/>
                                 <xsd:attribute name="date" type="xsd:string"/>
                                 <xsd:attribute name="amount" type="xsd:string"/>
                                 </xsd:complexType>
                              </xsd:element>
                           </xsd:sequence>
                        </xsd:complexType>
                     </xsd:element>
                     <xsd:element name="amount" type="xsd:string"/>
                  </xsd:sequence>
               <xsd:attribute name="id" type="xsd:string"/>
               </xsd:complexType>
            </xsd:element>
         </xsd:sequence>
      </xsd:complexType>
   </xsd:element>
</xsd:schema>
```

Ex 3: Step 4. Open the Data Sources and Target Schema

1. In the Data View Builder, choose File —> New Project to clear your desktop and reset all default values.

2. On the Builder Design —> Sources tab, click Data Views, and double-click on `AllOrders.xv` to open the schema for that data source.

3. Choose File —> Set Target Schema. Use the file browser to navigate to the Liquid Data Samples Server repository. Select `CustomerOrdersA.xsd` as the target schema. This target schema is displayed as a docked schema window on the right side of the design area.

Ex 3: Step 5. Map Source Nodes to Target to Project the Output

1. Drag and drop `[AllOrders]/ customers/customer/first_name` from the AllOrders source schema onto `[CustomerOrdersA.xsd]/customers/customer/first_name` in the target schema.

2. Drag and drop `[AllOrders]/ customers/customer/last_name` from AllOrders source schema onto `[CustomerOrdersA.xsd]/customers/customer/last_name` in the target schema.

Ex 3: Step 6. Create Two Query Parameters to be Provided at Query Runtime

Create two Query Parameter variables: `first_name` and `last_name`, that you can use to insert variable customer information when the query runs. Create both variables as type `xs:string`. Do this as follows:

1. On the Builder Toolbar, click Toolbox and then click Query Parameter.

2. From the Type drop-down menu, choose `xs:string`.

3. In Parameter Name field, enter `first_name` and click Add.

   The new parameter is displayed in the Query Parameters tree.

4. Repeat steps 2 and 3 to create the `last_name` variable.

   You should now see both parameters displayed in the Query Parameters tree.

Ex 3: Step 7. Assign the Query Parameters to Source Nodes

Assign the `first_name` and `last_name` Query Parameter variables to customer first name and last name nodes in the AllOrders data view as follows:

1. Drag and drop the `first_name` variable onto `[allOrders]/customers/customer/first_name` in the AllOrders source schema.

2. Drag and drop the `last_name` variable onto `[allOrders]/customers/customer/last_name` in the AllOrders source schema.

Ex 3: Step 8. Add the Count XQuery Function

Add the `count` XQuery function and specify the input and output as follows:

1. On the Builder Toolbar, click Toolbox and then click XQuery Functions.
2. Double-click on the `count` function (under Aggregate Functions)

   The `count` function window is displayed, showing input parameter *`srcval`* and output as some integer result.

Note: Create complex or aggregate functions only on the desktop by double-clicking as described in this step. Do not attempt to drag and drop them directly into the Conditions tab.

3. Drag and drop `[AllOrders]/customer/orders/order/date` from the AllOrders source schema onto `[count-Function]input/Parameters/srcval`.
4. Drag and drop `[count-Function]Output/result` to `[customerOrdersA.xsd]/customers/customer/amount` in the target schema.

   Note: Make sure to drag `result` onto the customer `amount` — the last node in the fully expanded schema tree; not onto the optional orders `amount?`

Ex 3: Step 9. Verify Mappings and Conditions

Your mappings should look like those shown in Figure 2-6.

Figure 2-6 Mappings for Example2: Aggregates

	Source	Target
1	[view:AllOrders]/customers/customer/first_name	[customerOrdersA.xsd]/customers/customer/first_name
2	[view:AllOrders]/customers/customer/last_name	[customerOrdersA.xsd]/customers/customer/last_name
3	[view:AllOrders]/customers/customer/orders/order/@date	[xf:count]/srcval
4	[xf:count]/result	[customerOrdersA.xsd]/customers/customer/amount
5		

Your Conditions should like those shown in Figure 2-7.

Figure 2-7 Conditions for Example 2: Aggregates

	Enabled	Condition
1	☒	(last_name eq [view:AllOrders]/customers/customer/last_name)
2	☒	(first_name eq [view:AllOrders]/customers/customer/first_name)
3		

Ex 3: Step 10. View the XQuery and Test by Running the Query

1. Click on the Test tab.

   The generated XQuery for this query is shown in the following code listing.

Listing 2-6 XQuery for Example 2: Aggregates

```
namespace view = "urn:views"
<customers>
   {
   for $view:AllOrders.customer_2 in view:AllOrders()/customers/customer
   let $srcval_3 :=
               for $view:AllOrders.orders_5 in $view:AllOrders.customer_2/orders
                   for $view:AllOrders.order_6 in $view:AllOrders.orders_5/order
                   return
                   xf:data($view:AllOrders.order_6/@date)
   where ($#last_name of type xs:string  eq
$view:AllOrders.customer_2/last_name)
     and ($#first_name of type xs:string  eq
$view:AllOrders.customer_2/first_name)
   return
   <customer>
      <first_name>{ xf:data($view:AllOrders.customer_2/first_name)
}</first_name>
      <last_name>{ xf:data($view:AllOrders.customer_2/last_name) }</last_name>
      <amount>{ cast as xs:string(xf:count($srcval_3)) }</amount>
   </customer>
   }
</customers>
```

2. In the Query Parameter panel on the Test tab, set the variable values as follows:
   - last_name

     (For last_name, `KAY_1` through `KAY_10` are available in the data source.)
   - first_name

     (For first_name, `JOHN_1` through `JOHN_10` are available in the data source.)
3. Click the Run query button to run the query against the data sources.

Ex 3: Step 11. Verify the Result

Running this query with `last_name` set to `KAY_1` and `first_name` set to `JOHN_1` produces the following XML query result.

Listing 2-7 Result for Example 2: Aggregates

```
<customers>
  <customer>
    <first_name>JOHN_1</first_name>
    <last_name>KAY_1</last_name>
    <amount>2</amount>
  </customer>
</customers>
```

Example 4: Date and Time Duration

Data View Builder supports a set of functions that operate on date and time. (For more information on date and time functions see "Date and Time Functions" in the *XQuery Reference Guide*.)

The Problem

Determine if a BroadBand customer has any open orders in the BroadBand database before a specified date.

The Solution

For each BroadBand order that matches the given Customer ID, you need to set these conditions:

- The order status is "OPEN"
- The ship date for a given *customer_id* is earlier than or equal to the date (*date1*) provided. (*customer_id* and *date1* are a variables that you define as query parameters to be submitted at query runtime).

To create the solution, follow these steps:

- View a Demo
- Ex 4: Step 1. Verify the Target Schema is Saved in Repository
- Ex 4: Step 2. Open Source and Target Schemas
- Ex 4: Step 3. Map Source to Target Nodes to Project the Output
- Ex 4: Step 4. Create Joins
- Ex 4: Step 5. Create Two Query Parameters for Customer ID and Date to be Provided at Query Runtime
- Ex 4: Step 6. Set a Condition Using the Customer ID
- Ex 4: Step 7. Set a Condition to Determine if Order Ship Date is Earlier or Equal to a Date Submitted at Query Runtime
- Ex 4: Step 8. Set a Condition to Include Only "Open" Orders in the Result
- Ex 4: Step 9. View the XQuery and Run the Query to Test it
- Ex 4: Step 9. Verify the Result

View a Demo

Date and Time Duration Demo... If you are looking at this documentation online, you can click the "Demo" button to see a viewlet demo showing how to build the conditions and create the mappings described in this example. This demo previews the steps described in detail in the following sections. The demo assumes you already have the target schema in the Liquid Data Samples Server repository.

Ex 4: Step 1. Verify the Target Schema is Saved in Repository

For this example, we will use a target schema called `customerLineItems.xsd`. This schema is available in the Liquid Data Samples Server repository.

`<WL_HOME>/samples/domains/liquiddata/ldrepository/schemas/`

Just in case you want to verify that you have the right schema file, the following code listing shows the XML for this schema.

Listing 2-8 XML Source for customerLineItems.xsd Target Schema File

```
<?xml version = "1.0" encoding = "UTF-8"?>

<xsd:schema xmlns:xsd = "http://www.w3.org/2001/XMLSchema">
   <xsd:element name = "customers">
      <xsd:complexType>
         <xsd:sequence>
            <xsd:element ref = "customer" minOccurs = "0" maxOccurs =
"unbounded"/>
         </xsd:sequence>
      </xsd:complexType>
   </xsd:element>
   <xsd:element name = "customer">
      <xsd:complexType>
         <xsd:sequence>
            <xsd:element ref = "first_name"/>
            <xsd:element ref = "last_name"/>
            <xsd:element ref = "orders" minOccurs = "0" maxOccurs = "unbounded"/>
         </xsd:sequence>
         <xsd:attribute name = "id" use = "required" type = "xsd:string"/>
      </xsd:complexType>
   </xsd:element>
   <xsd:element name = "first_name" type = "xsd:string"/>
   <xsd:element name = "last_name" type = "xsd:string"/>
   <xsd:element name = "orders">
      <xsd:complexType>
         <xsd:sequence>
```

```
            <xsd:element ref = "order" minOccurs = "0" maxOccurs = "unbounded"/>
         </xsd:sequence>
      </xsd:complexType>
   </xsd:element>
   <xsd:element name = "order">
      <xsd:complexType>
         <xsd:sequence>
            <xsd:element ref = "line_item" minOccurs = "0" maxOccurs =
"unbounded"/>
         </xsd:sequence>
         <xsd:attribute name = "id" use = "required" type = "xsd:string"/>
         <xsd:attribute name = "date" use = "required" type = "xsd:string"/>
         <xsd:attribute name = "amount" use = "required" type = "xsd:string"/>
      </xsd:complexType>
   </xsd:element>
   <xsd:element name = "line_item">
      <xsd:complexType>
         <xsd:attribute name = "id" use = "required" type = "xsd:string"/>
         <xsd:attribute name = "product" use = "required" type = "xsd:string"/>
         <xsd:attribute name = "status" use = "required" type = "xsd:string"/>
         <xsd:attribute name = "expected_ship_date" use = "required" type =
"xsd:string"/>
      </xsd:complexType>
   </xsd:element>
</xsd:schema>
```

Ex 4: Step 2. Open Source and Target Schemas

1. In the Data View Builder, choose File—>New Project to clear your desktop and reset all default values.
2. On the Builder Design—>Sources tab, click Relational Databases and open one data source:
   - Double-click on the `PB-BB` (BroadBand) relational database to open the schema for this data source.
3. Choose the menu option File—>Set Target Schema.

   Navigate to the Liquid Data Samples Server repository or to the location where you saved the `customerLineItems.xsd` schema. Choose `customerLineItems.xsd` and click Open.

   `customerLineItems.xsd` appears as the target schema.

   This target schema is displayed as a docked schema window on the right side of the design area.

4. Click the plus (+) sign (or right-mouse click and choose Expand) to expand the nodes in each source schema and in the target schema.

Ex 4: Step 3. Map Source to Target Nodes to Project the Output

Project the output values as follows.

1. Drag and drop `[PB-BB]/db/CUSTOMER/FIRST_NAME` from the source schema onto `[customerLineItems.xsd]/customers/customer/first_name` in the target schema.

2. Drag and drop `[PB-BB]/db/CUSTOMER/LAST_NAME` from the source schema onto `[customerLineItems.xsd]/customers/customer/last_name` in the target schema.

3. Drag and drop `[PB-BB]/db/CUSTOMER/CUSTOMER_ORDER_LINE_ITEM/LINE_ID` from the source schema onto `[customerLineItems.xsd]/customers/customer/orders/order/line_item/id` in the target schema (`id` is an *attribute* of line_item).

4. Drag and drop `[PB-BB]/db/CUSTOMER/CUSTOMER_ORDER_LINE_ITEM/PRODUCT_NAME` from the source schema onto `[customerLineItems.xsd]/customers/customer/orders/order/line_item/product` in the target schema (`product` is an *attribute* of line_item).

5. Drag and drop `[PB-BB]/db/CUSTOMER/CUSTOMER_ORDER_LINE_ITEM/STATUS` from the source schema `[customerLineItems.xsd]/customers/customer/orders/order/line_item/status` in the target schema (`status` is an *attribute* of line_item).

6. Drag and drop `[PB-BB]/db/CUSTOMER/CUSTOMER_ORDER_LINE_ITEM/EXPECTED_SHIP_DATE` from the source schema `[customerLineItems.xsd]/customers/customer/orders/order/line_item/expected_ship_date` in the target schema (`expected_ship_date` is an *attribute* of line_item).

At this point, the following mappings should be displayed on the Mappings tab. (Getting the mappings in the same order as shown is not as important as verifying that the relationships between source and target nodes are the same. The @ symbols indicate attributes.)

Source	Target
[PB-BB]/db/CUSTOMER/FIRST_NAME	[customerLineItems.xsd]/customers/customer/ first_name
[PB-BB]/db/CUSTOMER/LAST_NAME	[customerLineItems.xsd]/customers/customer/ last_name
[PB-BB]/db/CUSTOMER/CUSTOMER_ ORDER_LINE_ITEM/LINE_ID	[customerLineItems.xsd]/customers/customer/ orders/order/line_item/@id
[PB-BB]/db/CUSTOMER/CUSTOMER_ ORDER_LINE_ITEM/PRODUCT_NAME	[customerLineItems.xsd]/customers/customer/ orders/order/line_item/@product
[PB-BB]/db/CUSTOMER/CUSTOMER_ ORDER_LINE_ITEM/STATUS	[customerLineItems.xsd]/customers/customer/ orders/order/line_item/@status
[PB-BB]/db/CUSTOMER/CUSTOMER_ ORDER_LINE_ITEM/EXPECTED_SHIP_ DATE	[customerLineItems.xsd]/customers/customer/ orders/order/line_item/@expected_ship_ date

Ex 4: Step 4. Create Joins

Join customer with corresponding line-item data. This requires two joins, one to find the customer's Order IDs, and another that uses the Order IDs and finds the corresponding line-item information:

1. Drag and drop `[PB-BB]/db/CUSTOMER/CUSTOMER_ID` onto `[PB-BB]/db/CUSTOMER_ORDER/CUSTOMER_ID`
2. Drag and drop `[PB-BB]/db/CUSTOMER_ORDER/ORDER_ID` onto `[PB-BB]/db/CUSTOMER_ORDER_LINE_ITEM/ORDER_ID`

Ex 4: Step 5. Create Two Query Parameters for Customer ID and Date to be Provided at Query Runtime

Create two Query Parameter variables: *customer_id* and *date1*, that you can use to insert as variable values when the query runs. Create both variables as type `xs:string`. Do this as follows:

1. On the Builder Toolbar, click Toolbox and then click Query Parameter.
2. From the "Type" drop-down menu, choose `xs:string`.
3. In Parameter Name field, enter `customer_id` and click Add.

   The new parameter is displayed in the Query Parameters tree.
4. Repeat steps 2 and 3 to create the `date1` variable.

   You should now see both parameters displayed in the Query Parameters tree.

Ex 4: Step 6. Set a Condition Using the Customer ID

1. On the Builder Toolbar, click Toolbox and then click XQuery Functions.
2. Drag and drop the equals [`eq`] function (under Comparison operators) onto the next empty row in the Conditions tab.

   The Functions Editor appears, displaying placeholder variables for you to fill in.
3. On the Builder Toolbar, click on Query Parameter, then drag *customer_id* onto *anyValue1* onto the left side of the equation.
4. Drag `[PB-BB]/db/CUSTOMER/CUSTOMER_ID` onto the right side of the equation.

   The function should look like this:

   ```
   (customer_id eq [PB-BB]/db/CUSTOMER/CUSTOMER_ID)
   ```
5. Close the Functions Editor.

Ex 4: Step 7. Set a Condition to Determine if Order Ship Date is Earlier or Equal to a Date Submitted at Query Runtime

1. Click on XQuery Functions, and drag and drop the Operator function [`le`] (less than or equal) onto the next empty row on the Conditions tab.

   The Functions Editor pops up and displays a statement with placeholder variables for you to fill in.
2. Drag and drop `[PB-BB]/db/CUSTOMER_ORDER_LINE_ITEM/EXPECTED_SHIP_DATE` onto *anyValue1* on the left side of the equation.
3. Click on XQuery Functions, and drag and drop the `date-from-string-with-format` function (under Data and Time functions) onto *anyValue2* on the right side of the equation.

At this point, the expression in the Functions Editor should appear as:

```
([PB-BB]/db/CUSTOMER_ORDER_LINE_ITEM/EXPECTED_SHIP_DATE le
xfext:date-from-string-with-format(pattern,srcval))
```

4. Click Constants, enter the following in the String field:

   ```
   yyyy-MM-dd
   ```

   Now drag it (via the Constant icon next to the field) onto *pattern* (first placeholder parameter to the date function).

5. Click on Query Parameter, and drag and drop *date1* from the Query Parameters panel onto *srcval* (the second placeholder parameter to the date function).

   The completed expression should look like this:

```
([PB-BB]/db/CUSTOMER_ORDER_LINE_ITEM/EXPECTED_SHIP_DATE  le
xfext:date-from-string-with-format("yyyy-MM-dd",date1))
```

6. Close the Functions Editor.

   The condition you created is displayed on the Conditions tab in the Source column.

Ex 4: Step 8. Set a Condition to Include Only "Open" Orders in the Result

Set the second condition to an Open ORDER status.

1. Click on XQuery Functions, and drag and drop the Comparison operator [eq] (equal) function onto the Conditions tab.

   The Functions Editor pops up and displays a statement with placeholder variables for you to fill in.

2. For the left parameter (*anyValue1*), drag and drop `[PB-BB]/db/customer_order_line_item/status` on to *anyValue1*.

3. For the right parameter (*anyValue2*), create a constant String with a value of OPEN, and drop it (via the Constant icon next to the field) onto *anyValue2*.

   The completed expression should look like this:

   ```
   ([PB-BB]/db/CUSTOMER_ORDER_LINE_ITEM/STATUS eq "OPEN")
   ```

   Close the Functions Editor.

Ex 4: Step 9. View the XQuery and Run the Query to Test it

1. Click on the Test tab.

The generated XQuery for this query is shown in the following code listing.

Listing 2-9 XQuery for Example 3: Date and Time Duration

```
<customers>
   {
   for $PB_BB.CUSTOMER_1 in document("PB-BB")/db/CUSTOMER
   where ($#customer_id of type xs:string  eq  $PB_BB.CUSTOMER_1/CUSTOMER_ID)
   return
   <customer>
      <first_name>{ xf:data($PB_BB.CUSTOMER_1/FIRST_NAME) }</first_name>
      <last_name>{ xf:data($PB_BB.CUSTOMER_1/LAST_NAME) }</last_name>
      <orders>
         <order>
            {
            for $PB_BB.CUSTOMER_ORDER_LINE_ITEM_4 in
document("PB-BB")/db/CUSTOMER_ORDER_LINE_ITEM
            where xf:not(xf:empty(
               for $PB_BB.CUSTOMER_ORDER_5 in
document("PB-BB")/db/CUSTOMER_ORDER
               where ($PB_BB.CUSTOMER_1/CUSTOMER_ID  eq
$PB_BB.CUSTOMER_ORDER_5/CUSTOMER_ID)
                 and ($PB_BB.CUSTOMER_ORDER_5/ORDER_ID  eq
$PB_BB.CUSTOMER_ORDER_LINE_ITEM_4/ORDER_ID)
               return
               xf:true()))
              and ($PB_BB.CUSTOMER_ORDER_LINE_ITEM_4/STATUS  eq  "OPEN")
              and ($PB_BB.CUSTOMER_ORDER_LINE_ITEM_4/EXPECTED_SHIP_DATE  le
xfext:date-from-string-with-format("yyyy-MM-dd",$#date1 of type xs:string))
            return
            <line_item id={$PB_BB.CUSTOMER_ORDER_LINE_ITEM_4/LINE_ID}
product={$PB_BB.CUSTOMER_ORDER_LINE_ITEM_4/PRODUCT_NAME}
status={$PB_BB.CUSTOMER_ORDER_LINE_ITEM_4/STATUS}
expected_ship_date={$PB_BB.CUSTOMER_ORDER_LINE_ITEM_4/EXPECTED_SHIP_DATE} />
            }
         </order>
      </orders>
   </customer>
   }
</customers>
```

2. In the Query Parameter panel on the Test tab, set the variable values for customer_id and date1 to submit to the query when the query runs.

For example:

- customer_id: CUSTOMER_1 (CUSTOMER_1 through CUSTOMER_10 are available in the data source.)
- date1: 2002-08-01 (You can enter any date in the form *yyyy-MM-dd*.)

3. Click the Run query button to run the query against the data sources.

Ex 4: Step 9. Verify the Result

Running this query with `customer_id` set to CUSTOMER_1 and `date1` set to 2002-08-01 produces the following XML query result.

Listing 2-10 Result for Example 3: Date and Time Duration

```
<customers>
  <customer>
    <first_name>JOHN_B_1</first_name>
    <last_name>KAY_1</last_name>
    <orders>
      <order>
        <line_item id="LINE_ID_1" product="RBBC01" status="OPEN"
expected_ship_date="2002-03-06"/>
        <line_item id="LINE_ID_3" product="BN16" status="OPEN"
expected_ship_date="2002-03-06"/>
        <line_item id="LINE_ID_5" product="CS100" status="OPEN"
expected_ship_date="2002-03-06"/>
        <line_item id="LINE_ID_1" product="RBBC01" status="OPEN"
expected_ship_date="2002-03-06"/>
        <line_item id="LINE_ID_3" product="BN16" status="OPEN"
expected_ship_date="2002-03-06"/>
        <line_item id="LINE_ID_5" product="CS100" status="OPEN"
expected_ship_date="2002-03-06"/>
      </order>
    </orders>
  </customer>
</customers>
```

Example 5: Union

A union query is equivalent to concatenating two or more subordinate queries, and pooling the query results into the same output. There are two important rules for a union query.

- Each subordinate query produces a result directed at a repeatable target schema node that is not shared (parent or child) with any other subordinate query target.
- You cannot specify any conditions across these subordinate queries.

The Problem

For any BroadBand Customer ID, list any BroadBand and Wireless orders. Assume the Customer IDs match across databases.

The Solution

This query requests a union of BroadBand orders and Wireless orders. Remember that a union retrieves data from multiple sources, such as the BroadBand and Wireless databases, but there are no conditions for the query. If you specify any condition, such as matching order dates, then you are creating a join query. In this example, you need a target schema that contains a repeatable list of Customer IDs, and within that list, a repeatable list of orders. Then you will clone the orders element, using one element for BroadBand orders and the other element for Wireless orders.

To create the solution, follow these steps:

- View a Demo
- Ex 5: Step 1. Verify the Target Schema is Saved in Repository
- Ex 5: Step 2. Open Source and Target Schemas
- Ex 5: Step 3. Clone the Orders Element of the Target Schema
- Ex 5: Step 4. Create a Query Parameter for a Customer ID
- Ex 5: Step 5. Assign a Query Parameters
- Ex 5: Step 6. Define Source Relationships
- Ex 5: Step 7. Project the Output to the Target Schema
- Ex 5: Step 8. View, then Run the Query
- Ex 5: Step 9. Verify the Result

View a Demo

Union Demo... If you are looking at this documentation online, you can click the "Demo" button to see a viewlet demo showing how to build the conditions and create the mappings described in this example. This demo previews the steps described in detail in the following sections. The demo assumes you already have the target schema in the Liquid Data Samples Server repository.

Ex 5: Step 1. Verify the Target Schema is Saved in Repository

For this example, we will use a target schema called `unionOrders.xsd`. This schema is available in the Liquid Data Samples Server repository. The path to the schemas folder is:

```
<WL_HOME>/samples/domains/liquiddata/ldrepository/schemas/
```

Just in case you want to verify that you have the right schema file, the following code listing shows the XML for this schema.

Listing 2-11 XML Source for unionOrders.xsd Target Schema File

```
<?xml version = "1.0" encoding = "UTF-8"?>
<!--Generated by Data View Builder 1.1. Conforms to w3c
http://www.w3.org/2001/XMLSchema-->
<xsd:schema xmlns:xsd = "http://www.w3.org/2001/XMLSchema"  >
   <xsd:element name="customers">
      <xsd:complexType>
         <xsd:sequence>
            <xsd:element name="customer" minOccurs="0" maxOccurs="unbounded">
               <xsd:complexType>
                  <xsd:sequence>
                     <xsd:element name="first_name" type="xsd:string"/>
                     <xsd:element name="last_name" type="xsd:string"/>
                     <xsd:element name="state" type="xsd:string"/>
                     <xsd:element name="orders" minOccurs="0"
maxOccurs="unbounded">
                        <xsd:complexType>
                           <xsd:sequence>
                              <xsd:element name="order" minOccurs="0"
maxOccurs="unbounded">
                                 <xsd:complexType>
                                    <xsd:sequence>
                                       <xsd:element name="date"
type="xsd:string"/>
                                       <xsd:element name="amount"
type="xsd:decimal"/>
                                    </xsd:sequence>
```

```
                                    </xsd:complexType>
                                </xsd:element>
                            </xsd:sequence>
                        </xsd:complexType>
                    </xsd:element>
                </xsd:sequence>
            </xsd:complexType>
        </xsd:element>
    </xsd:sequence>
</xsd:complexType>
</xsd:element>
</xsd:schema>
```

Ex 5: Step 2. Open Source and Target Schemas

1. In the Data View Builder, choose File —> New Project to clear your desktop and reset all default values.
2. On the Builder Toolbar Design —> Sources tab, click Relational Databases and open two data sources:
   - Double-click on the `PB-WL` (Wireless) relational database to open the schema for this data source.
   - Double-click on the `PB-BB` (BroadBand) relational database to open the schema for this data source.
3. Choose the menu option File —> Set Target Schema.
4. Navigate to the Liquid Data Samples Server repository. Choose `unionOrders.xsd` and click Open.

   `unionOrders.xsd` appears as the target schema.
5. Click the plus (+) sign (or right-mouse click and choose Expand) to expand the nodes in each source schema and in the target schema.

Ex 5: Step 3. Clone the Orders Element of the Target Schema

1. In the Data View Builder, select the Orders element of the target schema and right-mouse click. The Orders element is a child of the Customers element and has a child called Order.
2. Choose Clone from the right-mouse menu.

The Cloned complex element labeled `Orders(2)` appears. The name of the original complex element is changed to `Orders(1)`.

Ex 5: Step 4. Create a Query Parameter for a Customer ID

Create a Query Parameter variable, *customer_id*, that you can use to insert as a variable for a BroadBand customer ID value when the query runs. To create this parameter, do the following:

1. On the Builder Toolbar, click Toolbox and then click Query Parameter.
2. From the Type drop-down menu, choose `xs:string`.
3. In Parameter Name field, enter `customer_id` and click Create.

   The new parameter is displayed in the Query Parameters tree.

Ex 5: Step 5. Assign a Query Parameters

Assign the query parameter `customer_id` to the BroadBand customer ID by dragging and dropping query parameter `customer_id` to the `[PB-BB]/db/CUSTOMER/CUSTOMER_ID` element.

Ex 5: Step 6. Define Source Relationships

1. Within PB-BB, join the BroadBand Customer ID to the Order Customer ID.

   Drag and drop `[PB-BB]/db/CUSTOMER/CUSTOMER_ID` onto BroadBand `[PB-BB]/db/CUSTOMER_ORDER/CUSTOMER_ID`

2. Join the BroadBand customer ID from the BroadBand Customer table with the Wireless customer ID from the Wireless Customer Order table as follows:

   Drag and drop `[PB-BB]/db/CUSTOMER/CUSTOMER_ID` onto `[PB-WL]/db/CUSTOMER_ORDER/CUSTOMER_ID`

Ex 5: Step 7. Project the Output to the Target Schema

1. Project the BroadBand order information.
   - Drag and drop `[PB-BB]/db/CUSTOMER_ORDER/TOTAL_ORDER_AMOUNT` onto `[UnionOrders.xsd]/customers/customer/orders(1)/order/amount`
   - Drag and drop `[PB-BB]/db/CUSTOMER_ORDER/ORDER_DATE` onto `[UnionOrders.xsd]/customers/customer/orders(1)/order/date`
2. Project the Wireless [PB-WL] order information.

- Drag and drop `[PB-WL]/db/CUSTOMER_ORDER/TOTAL_ORDER_AMOUNT` onto `[UnionOrders.xsd]/customers/customer/orders(2)/order/amount`
- Drag and drop `[PB-WL]/db/CUSTOMER_ORDER/ORDER_DATE` onto `[UnionOrders.xsd]/customers/customer/orders(2)/order/date`

3. Project the BroadBand user information.
   - Drag and drop `[PB-BB]/db/CUSTOMER/FIRST_NAME` onto `[unionOrders.xsd]/customers/customer/first_name`
   - Drag and drop `[PB-BB]/db/CUSTOMER/LAST_NAME` onto `[unionOrders.xsd]/customers/customer/last_name`
   - Drag and drop `[PB-BB]/db/CUSTOMER/STATE` onto `[unionOrders.xsd]/customers/customer/state`

Ex 5: Step 8. View, then Run the Query

1. Click on the Test tab.

   The generated XQuery for this query is shown in the following code listing.

Listing 2-12 XQuery for Example 4: Union

```
<customers>
  {
  for $PB_BB.CUSTOMER_1 in document("PB-BB")/db/CUSTOMER
  where ($#customer_id of type xs:string  eq  $PB_BB.CUSTOMER_1/CUSTOMER_ID)
  return
  <customer>
    <orders>
       {
       for $PB_BB.CUSTOMER_ORDER_6 in document("PB-BB")/db/CUSTOMER_ORDER
       where ($PB_BB.CUSTOMER_1/CUSTOMER_ID  eq
$PB_BB.CUSTOMER_ORDER_6/CUSTOMER_ID)
       return
       <order>
          <date>{ xf:data($PB_BB.CUSTOMER_ORDER_6/ORDER_DATE) }</date>
          <amount>{ xf:data($PB_BB.CUSTOMER_ORDER_6/TOTAL_ORDER_AMOUNT)
}</amount>
       </order>
       }
    </orders>
    <orders>
       {
       for $PB_WL.CUSTOMER_ORDER_9 in document("PB-WL")/db/CUSTOMER_ORDER
```

```
      where ($PB_BB.CUSTOMER_1/CUSTOMER_ID  eq
$PB_WL.CUSTOMER_ORDER_9/CUSTOMER_ID)
      return
      <order>
         <date>{ xf:data($PB_WL.CUSTOMER_ORDER_9/ORDER_DATE) }</date>
         <amount>{ xf:data($PB_WL.CUSTOMER_ORDER_9/TOTAL_ORDER_AMOUNT)
}</amount>
      </order>
      }
   </orders>
   <custID>{ xf:data($PB_BB.CUSTOMER_1/CUSTOMER_ID) }</custID>
  </customer>
  }

</customers>
```

2. In the Query Parameter panel, click into the cell under "Value" and enter a value for `customer_id`. (CUSTOMER_1 through CUSTOMER_10 are available to try.)
3. Click the Run query arrow to run the query against the data sources.

Ex 5: Step 9. Verify the Result

Querying these data sources as described in this example produces an XML query result similar to that shown in the following code listing where CUSTOMER_4 was used as the query parameter value for `customer_id`.

Listing 2-13 Result for Example 4: Union

```
<customers>
  <customer>
    <first_name>JOHN_B_4</first_name>
    <last_name>KAY_4</last_name>
    <state>NV</state>
    <orders>
      <order>
        <date>2002-03-06-08:00</date>
        <amount>1000</amount>
      </order>
      <order>
```

```
    <date>2002-03-06-08:00</date>
    <amount>1500</amount>
  </order>
  <order>
    <date>2002-03-06-08:00</date>
    <amount>2000</amount>
  </order>
  <order>
    <date>2002-03-06-08:00</date>
    <amount>2500</amount>
  </order>
  <order>
    <date>2002-03-06-08:00</date>
    <amount>3000</amount>
  </order>
</orders>
<orders>
  <order>
    <date>2002-03-06-08:00</date>
    <amount>1000</amount>
  </order>
  <order>
    <date>2002-03-06-08:00</date>
    <amount>2000</amount>
  </order>
  <order>
    <date>2002-03-06-08:00</date>
    <amount>4000</amount>
  </order>
  <order>
    <date>2002-03-06-08:00</date>
    <amount>5000</amount>
  </order>
  <order>
    <date>2002-03-06-08:00</date>
    <amount>10000</amount>
  </order>
</orders>
```

```
  </customer>
</customers>
```

Example 6: Minus

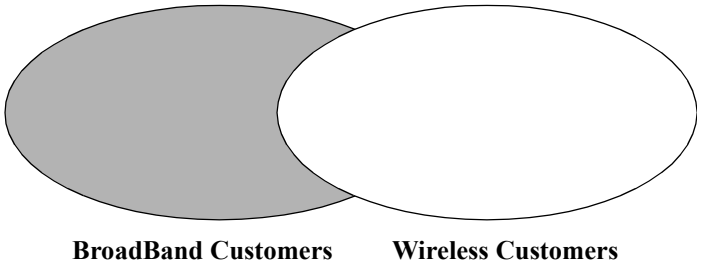
A minus relationship (A minus B) returns all instances of some named value that are in A but not in B. There is no explicit minus operation in the XQuery language or Data View Builder; however, a simple compare and count technique can be used. For example: for each instance of the named value in A, count all matching instances in B; if the count is zero, that means there are no matches, and the query therefore returns the instance from A.

The Problem

Find all customers that are BroadBand customers, but not Wireless customers. Assume that Customer IDs match across databases.

The shaded area in Figure 2-8 represents the BroadBand customers who are not Wireless customers.

Figure 2-8 BroadBand and Wireless Customers



The Solution

If a customer has only a BroadBand account, then a join across the BroadBand and Wireless databases on that Customer ID produces an empty result. We can take advantage of that fact by counting the number of instances produced by the join. If the number is zero, then the Customer ID represents a BroadBand-only customer.

To create the solution, follow these steps:

- View a Demo
- Ex 6: Step 1. Verify the Target Schema is Saved in Repository
- Ex 6: Step 2. Open Source and Target Schemas
- Ex 6: Step 3. Find BroadBand and Wireless Customers with the Same Customer ID
- Ex 6: Step 4. Find the Count of the Wireless Customers

View a Demo

Minus Demo... If you are looking at this documentation online, you can click the "Demo" button to see a viewlet demo showing how to build the conditions and create the mappings described in this example. This demo previews the steps described in detail in the following sections. The demo assumes you already have the target schema in the Liquid Data Samples Server repository.

Ex 6: Step 1. Verify the Target Schema is Saved in Repository

For this example, we will use a target schema called `minus.xsd`. This schema is available in the Liquid Data Samples Server repository. The path to the schemas folder is:

```
<WL_HOME>/samples/domains/liquiddata/ldrepository/schemas/
```

Just in case you want to verify that you have the right schema file, the following code listing shows the XML for this schema.

Listing 2-14 XML Source for minus.xsd Target Schema File

```
<?xml version = "1.0" encoding = "UTF-8"?>

<xsd:schema xmlns:xsd = "http://www.w3.org/2001/XMLSchema">
   <xsd:element name="results">
      <xsd:complexType>
         <xsd:sequence>
            <xsd:element name="CUSTOMER" minOccurs="1" maxOccurs="unbounded">
               <xsd:complexType>
                  <xsd:sequence>
                     <xsd:element name="FIRST_NAME" type="xsd:string"/>
                     <xsd:element name="LAST_NAME" type="xsd:string"/>
                  </xsd:sequence>
               </xsd:complexType>
            </xsd:element>
         </xsd:sequence>
      </xsd:complexType>
   </xsd:element>
</xsd:schema>
```

Ex 6: Step 2. Open Source and Target Schemas

1. In the Data View Builder, choose File—>New Project to clear your desktop and reset all default values.
2. On the Builder Toolbar Design—>Sources tab, click Relational Databases and open two data sources:
   - Double-click on the `PB-WL` (Wireless) relational database to open the schema for this data source.
   - Double-click on the `PB-BB` (BroadBand) relational database to open the schema for this data source.
3. Choose the menu option File —> Set Target Schema.

   Navigate to the Liquid Data Samples Server repository or to the location where you saved the `minus.xsd` schema. Choose `minus.xsd` and click Open.

   `minus.xsd` appears as the target schema.

   This target schema is displayed as a docked schema window on the right side of the design area.
4. Click the plus (+) sign (or right-mouse click and choose Expand) to expand the nodes in each source schema and in the target schema.

Ex 6: Step 3. Find BroadBand and Wireless Customers with the Same Customer ID

- Drag and drop `[PB-BB]/db/CUSTOMER/CUSTOMER_ID` onto `[PB-WL]/db/CUSTOMER/CUSTOMER_ID` to join the BroadBand CUSTOMER_ID and the Wireless CUSTOMER_ID.

Ex 6: Step 4. Find the Count of the Wireless Customers

1. On the Builder Toolbar Design —> Toolbox tab, click XQuery Functions and double-click on the `xf:count` function (under Aggregate functions) to open it.
2. Drag and drop the `[PB-WL]/db/CUSTOMER/CUSTOMER_ID` onto the input of the `xf:count` function.

Ex 6: Step 5. Set a Condition that Specifies the Output of “count” is Zero

1. Click on the Conditions tab.

2. Drag and drop the `eq` (equal) function (in the XQuery functions Comparison operators folder) onto the next empty row under Conditions on the Conditions tab.

   The Functions Editor is displayed.

3. For the first parameter, drop `xf:count/result` onto `anyValue1`.

4. For the second parameter, create a Number constant, set it to `0` and drop it on `anyValue2`.

   Note: To create the Number constant, on Builder —> Toolbox tab, click Constants, enter 0 in the Number field, and drag the Constant icon next to that field onto `anyValue2` in the equation in the Functions Editor.

   The equality condition should look like this:

   ```
   ([xf:count]/result eq 0)
   ```

   Close the Functions Editor.

5. Project the BroadBand customers to the target results.

   – Drag and drop `[PB-BB]/db/CUSTOMER/FIRST_NAME` onto `[minus.xsd]/results/CUSTOMER/FIRST_NAME`

   – Drag and drop `[PB-BB]/db/CUSTOMER/LAST_NAME` onto `[minus.xsd]/results/CUSTOMER/LAST_NAME`

Ex 6: Step 6. View the XQuery and Run the Query to Test it

1. Click on the Test tab.

   The generated XQuery for this query is shown in the following code listing.

Listing 2-15 XQuery for Example 5: Minus

```
<results>
   {
   for $PB_BB.CUSTOMER_1 in document("PB-BB")/db/CUSTOMER
   let $srcval_2 :=
                   for $PB_WL.CUSTOMER_3 in document("PB-WL")/db/CUSTOMER
                   where ($PB_BB.CUSTOMER_1/CUSTOMER_ID  eq
$PB_WL.CUSTOMER_3/CUSTOMER_ID)
                   return
                   xf:data($PB_WL.CUSTOMER_3/CUSTOMER_ID)
   let $xf:count_4 := xf:count($srcval_2)
   where ($xf:count_4  eq  0)
   return
   <CUSTOMER>
      <FIRST_NAME>{ xf:data($PB_BB.CUSTOMER_1/FIRST_NAME) }</FIRST_NAME>
      <LAST_NAME>{ xf:data($PB_BB.CUSTOMER_1/LAST_NAME) }</LAST_NAME>
   </CUSTOMER>
   }
</results>
```

2. Click the “Run query” button to run the query against the data sources.

Ex 6: Step 7. Verify the Result

When you run this query on the sample data sources as described here, the result will be one record because the sample BroadBand data source has one customer record that is different from the Wireless customer records.

Listing 2-16 Result for Example 5: Minus

```
<results>
  <CUSTOMER>
    <FIRST_NAME>JOHN</FIRST_NAME>
    <LAST_NAME>PARKER</LAST_NAME>
```

```
    </CUSTOMER>
  </results>
```

Example 7: Complex Parameter Type (CPT)

The Complex Parameter Type Cookbook example shows how to use Liquid Data to create an integrated view that connects two enterprise information systems: a database and an in-flight XML data source using a complex parameter type (CPT). A query that uses both data sources determines whether the customer has sufficient credit for the incoming order to be processed.

The Problem

The company receives dozens of electronically transmitted orders daily and needs to quickly respond to its field office if an order cannot be accepted because a customer has exceeded their credit limit. The credit limit and amount of outstanding orders is known to the system. The quantity and price of the items being ordered is supplied in real-time along with the order.

The Solution

The company develops a complex parameter type (CPT) that models the incoming purchase order as an XML schema and sets a simple `orderLimit` parameter that an operator can change whenever the query is run. The query also calculates the total amount outstanding of current orders and the total amount of the incoming order. The objective is to accept orders if the total amount of both outstanding and incoming orders is within the order limit. Otherwise, the order is rejected.

To recreate the solution, follow these steps:

- Ex 7: Step 10: Test If Open Orders + New Orders Exceeds the Order Limit
- Ex 7: Step 11: Determine If the Order is Accepted or Rejected
- Ex 7: Step 12: View the XQuery
- Ex 7: Step 13. Run the XQuery to Verify the Result

Note: The implementation details of the Complex Parameter Type demo, the DB-COCPT sample, and the CPT cookbook example vary slightly.

View a Demo

Complex Parameter Type (CPT) Demo. If you are looking at this documentation online, you can click the "Demo" button to see a viewlet demo showing how to build the conditions and create the mappings described in this example.

Ex 7: Step 1. Verify the Availability of Schemas and Sample Data Stream

In creating the DB-CPTCO sample query, we use the following files that are installed with Liquid Data samples.

From the Liquid Data Samples Server repository schema directory:

- **BroadBand database schema.** The path to this file is:

```
<WL_HOME>/samples/domains/liquiddata/ldrepository/schemas/broadbandp.sql
```

- **Complex parameter type schema.** The path to this file is:

```
<WL_HOME>/samples/domains/liquiddata/ldrepository/schemas/coCptSample2.xsd
```

- **Target schema.** The path to this file is:

```
<WL_HOME>/samples/domains/liquiddata/ldrepository/schemas/COCPTSampleTarget-Schema.xsd
```

- **CPT sample XML stream.** The path to this file is:

```
<WL_HOME>/samples/domains/liquiddata/ldrepository/xml_files/coCPTsample2.xml
```

If you want to refer to the sample DB-CPTCO project, it is installed as the following file:

```
<WL_HOME>/<LD_HOME>/liquiddata/buildquery/db-cptco/coCPTSample.qpr
```

For reference purposes, code listings for several of the XML files used in this example appear below:

Listing 2-17 DB-CPTCO Sample CPT Schema (`coCptSample2.xsd`)

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema targetNamespace="urn:schemas-bea-com:ld-cocpt"
                 xmlns:cocpt="urn:schemas-bea-com:ld-cocpt"
                 xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <xsd:element name="CustOrder">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element name="CUSTOMER_ORDER" minOccurs="0" maxOccurs="unbounded">
          <xsd:complexType>
            <xsd:sequence>
              <xsd:element name="CUSTOMER_ID" type="xsd:string"/>
              <xsd:element name=

"NEW_ORDER_LINE_ITEM"type="cocpt:NEW_ORDER_LINE_ITEMType"
                       minOccurs="0" maxOccurs="unbounded"/>
            </xsd:sequence>
          </xsd:complexType>
        </xsd:element>
      </xsd:sequence>
    </xsd:complexType>
  </xsd:element>
  <xsd:complexType name="NEW_ORDER_LINE_ITEMType">
    <xsd:sequence>
      <xsd:element name="PRODUCT_NAME" type="xsd:string"/>
      <xsd:element name="QUANTITY" type="xsd:decimal"/>
      <xsd:element name="PRICE" type="xsd:decimal"/>
    </xsd:sequence>
  </xsd:complexType>
</xsd:schema>
```

Listing 2-18 DB-CPTCO Target Schema (`COCPTSampleTargetSchema.xsd`)

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema targetNamespace="urn:schemas-bea-com:ld-cocpt"
xmlns:cocpt="urn:schemas-bea-com:ld-cocpt"
xmlns:xsd="http://www.w3.org/2001/XMLSchema">
    <xsd:element name="CustOrder">
        <xsd:complexType>
            <xsd:sequence>
                <xsd:element name="CUSTOMER_ORDER" minOccurs="0"
maxOccurs="unbounded">
                    <xsd:complexType>
                        <xsd:sequence>
```

```
                                <xsd:element name="CUSTOMER_ID"
type="xsd:string"/>
                                <xsd:element name="NEW_ORDER_LINE_ITEM"
type="cocpt:NEW_ORDER_LINE_ITEMType" minOccurs="0" maxOccurs="unbounded"/>
                            </xsd:sequence>
                        </xsd:complexType>
                    </xsd:element>
                </xsd:sequence>
            </xsd:complexType>
        </xsd:element>
        <xsd:complexType name="NEW_ORDER_LINE_ITEMType">
            <xsd:sequence>
                <xsd:element name="PRODUCT_NAME" type="xsd:string"/>
                <xsd:element name="QUANTITY" type="xsd:decimal"/>
                <xsd:element name="PRICE" type="xsd:decimal"/>
            </xsd:sequence>
        </xsd:complexType>
</xsd:schema>
```

Listing 2-19 DB-CPTCO Sample XML Data Stream (`coCptSample2.xml`)

```
<?xml version="1.0" encoding="UTF-8"?>
<cocpt:CustOrder xmlns:cocpt="urn:schemas-bea-com:ld-cocpt"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="urn:schemas-bea-com:ld-cocpt
coCptSample2.xsd">
<CUSTOMER_ORDER>
<CUSTOMER_ID>CUSTOMER_1</CUSTOMER_ID>
<NEW_ORDER_LINE_ITEM>
<PRODUCT_NAME>RBBC01</PRODUCT_NAME>
<QUANTITY>1000</QUANTITY>
<PRICE>20</PRICE>
</NEW_ORDER_LINE_ITEM>
<NEW_ORDER_LINE_ITEM>
<PRODUCT_NAME>CS2610</PRODUCT_NAME>
<QUANTITY>1000</QUANTITY>
<PRICE>20</PRICE>
</NEW_ORDER_LINE_ITEM>
</CUSTOMER_ORDER>
</cocpt:CustOrder>
```

You may also want to examine the CO-CPTSAMPLE definition in the WLS Administration Console for the Samples server you are using.

1. Login to the Administration Server. See Start the WLS Administration Console in the Getting Started guide for details.
2. In the left pane, click the Liquid Data node.
3. In the right pane, click the Complex Parameter Types tab and click on CO-CPTSAMPLE.

See the section "Creating a Complex Parameter Type" for details.

Ex 7: Step 2. Open the Target Schema and CO-CPTSAMPLE CPT

1. In the Data View Builder, choose `File—>New Project`.
2. On the Design tab, on the Builder Toolbar, click the Toolbox tab, then click Complex Parameter Type. The `CO-CPTSAMPLE` complex parameter type is listed.
3. Double-click on `CO-CPTSAMPLE` to open the CPT schema.
4. Choose `File -> Set Target Schema`. Browse to the Liquid Data Samples Server repository schema directory.
5. Select the following file as the target schema:

```
<WL_HOME>/samples/domains/liquiddata/ldrepository/schemas/COCPTSampleTargetSch
ema.xsd
```

In the target schema window on the right side of the design area, right-click on the top element expand the target schema.

Ex: 7: Step 3. Create an orderLimit Query Parameter

Since credit limits vary from customer to customer, it is convenient to have an order limit query parameter that can be changed whenever a query is run.

1. In the Data View Builder select `Design—>Toolbox`.
2. Click on the Query Parameter tab.
3. Enter `orderLimit` as a parameter name.
4. Select `xs:decimal` as the parameter type.
5. Click Create.
6. Drag and drop the `orderLimit` parameter icon onto the target schema `[COCPTSampleTargetSchema.xsd]/cocpt:CustOrderStatus/CUSTOMER/CUSTOMER_ORDER/TOTAL_OPEN_ORDERLIMIT`

Ex 7: Step 4. Save the Project

You can save a project at any time. To initially create a project, use `File—>Save Project As`. Use the file browser to choose a location and project name (we use `myCoCPT`).

Ex 7: Step 5. Test Access to the Complex Parameter Source

Follow these steps to verify access to the CPT data source:

1. Drag and drop output from:

   `[CO-CPTSAMPLE]/cocpt:CustOrder/CUSTOMER_ORDER/CUSTOMER_ID`

   onto the target schema:

   `[COCPTSampleTargetSchema.xsd]/cocpt:CustOrderStatus/cocpt:CUSTOMER/cocpt:CUSTOMER_ID.`

2. Click the Test tab.

3. In the lower-left pane of the Data View Builder (Test mode), click in the Values area under Query Parameters to the right of the CPT name (`CO-CPTSAMPLE`).

4. Navigate to the XML data file associated with the CO-CPTSAMPLE complex parameter type.

   ```
   <WL_HOME>/samples/domains/liquiddata/ldrepository/xml_files/coCptSample2.xml
   ```

5. Enter an `orderLimit` value such as `200000`.

6. Now you can click the Run arrow to execute a preliminary query. The following result shows that your CPT is successfully retrieving from the XML file data:

Listing 2-20 Interim Results (1) from the CPT Example Query

```
<prefix1:CustOrderStatus xmlns:prefix1="urn:schemas-bea-com:ld-cocpt">
  <prefix1:CUSTOMER>
    <prefix1:CUSTOMER_ID>CUSTOMER_1</prefix1:CUSTOMER_ID>
    <prefix1:CUSTOMER_ORDER>
      <prefix1:TOTAL_OPEN_ORDERLIMIT>200000</prefix1:TOTAL_OPEN_ORDERLIMIT>
    </prefix1:CUSTOMER_ORDER>
  </prefix1:CUSTOMER>
</prefix1:CustOrderStatus>
```

7. Return to the Toolbar Design mode.

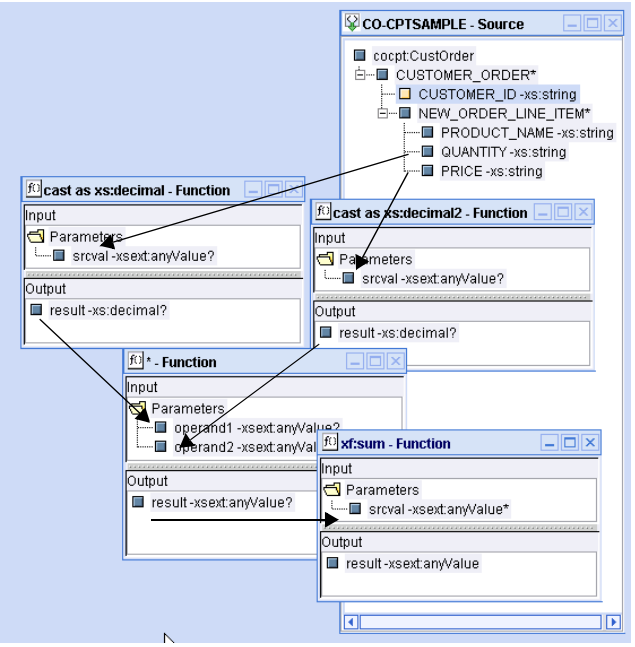
In the case of this data source, the customer identification is provided so there is no need to create a customer ID query parameter.

Ex 7: Step 6: Determine the Total Amount of New Orders

Since all runtime source items from a CPT are treated as character strings, any data items from the CO-CPTSAMPLE data source must first be cast appropriately. Then quantities and prices are multiplied together. The sum of the products of quantity and prices is the total amount of new orders.

1. Click XQuery Functions.
2. Drag and drop a multiply `[*]` function into the design area (from the Numeric operators group).
3. Drag and drop a sum Aggregate function (xf:sum) into the design area.

Figure 2-9 XQuery Functions Used to Calculate Total New Orders in Data Stream



4. Drag and drop the output result of `xs:decimal` to one side of the multiply equation and the output result of `xs:decimal2` to the other.

5. Drag and drop `[CO-CPTSAMPLE]cocpt:CustOrder/CUSTOMER_ORDER/NEW_ORDER_LINE_ITEM/QUANTITY` to one side of the multiply equation.
6. Drag and drop `[CO-CPTSAMPLE]cocpt:CustOrder/CUSTOMER_ORDER/NEW_ORDER_LINE_ITEM/PRICE` to the other side of the multiply equation.
7. Drag the `[*] function` output result to the input parameter of `xf:sum`. This gives you the total order amount in the CPT data source.
8. Drag and drop output from `xf:sum` onto the target schema `[COCPTSampleTargetSchema.xsd]/cocpt:CustOrderStatus/CUSTOMER/CUSTOMER_ORDER/NEW_ORDER_TOTAL_AMOUNT`
9. Rerun your query. A new order total of `40000` appears.
10. Return to Design mode.

Ex 7: Step 7. Create the Necessary Joins and Mappings to the Target Schema

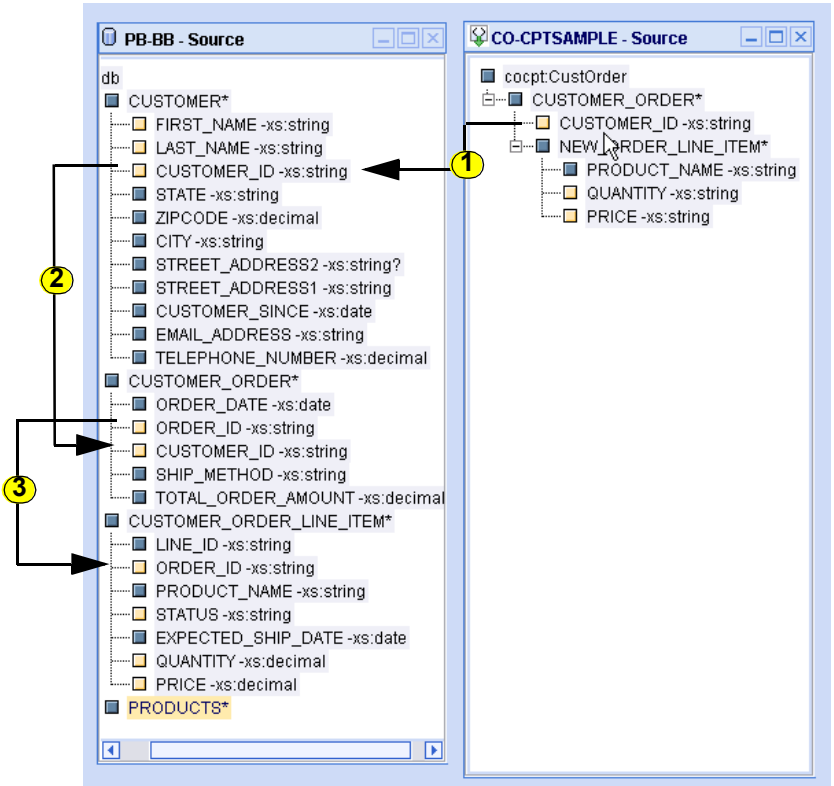
Move the PB-BB relational source schema onto the design area.

1. On the Builder Toolbar Design—>Sources tab, click Relational Databases, and double-click on `PB-BB` to open the schema for the BroadBand sample data source.
2. Expand the `PB-BB` schema.

Create the necessary joins to allow us to fetch the line items for a particular order for a particular customer.

1. Drag and drop `[CO-CPTSAMPLE]/cocpt:CustOrder/CUSTOMER_ORDER/CUSTOMER_ID` onto `[PB-BB]/db/CUSTOMER/CUSTOMER_ID`
2. Drag and drop `[PB-BB]/db/CUSTOMER/CUSTOMER_ID` onto the `[PB-BB]/db/CUSTOMER_ORDER/CUSTOMER_ID`
3. Drag and drop `[PB-BB]/db/CUSTOMER_ORDER/ORDER_ID` onto `[PB-BB]/db/CUSTOMER_ORDER_LINE_ITEM/ORDER_ID`

Figure 2-10 Setting Joins Between CPT and Relational Data Source



Next, project `FIRST_NAME` and `LAST_NAME` elements onto the target schema.

1. Drag and drop `[PB-BB]/db/CUSTOMER/FIRST_NAME` onto the target schema `[cocptsampletarget-schema]/cocptCustOrderStatus/CUSTOMER/FIRST_NAME`
2. Drag and drop `[PB-BB]/db/CUSTOMER/LAST_NAME` onto the target schema `[cocptsampletarget-schema]/cocptCustOrderStatus/CUSTOMER/LAST_NAME`

Note: In this version of the CO-CPTSAMPLE, automatic scoping is used. See "Understanding Condition Scoping" in *Building Queries and Data Views* for more information.

Although your query is not complete, you can test run it again.

Listing 2-21 Interim Results (2) from CPT Example Query

```
<prefix1:CustOrderStatus xmlns:prefix1="urn:schemas-bea-com:ld-cocpt">
  <prefix1:CUSTOMER>
    <prefix1:FIRST_NAME>JOHN_B_1</prefix1:FIRST_NAME>
    <prefix1:LAST_NAME>KAY_1</prefix1:LAST_NAME>
    <prefix1:CUSTOMER_ID>CUSTOMER_1</prefix1:CUSTOMER_ID>
    <prefix1:CUSTOMER_ORDER>
      <prefix1:NEWORDER_TOTAL_AMOUNT>40000</prefix1:NEWORDER_TOTAL_AMOUNT>
     <prefix1:TOTAL_OPEN_ORDERLIMIT>200000</prefix1:TOTAL_OPEN_ORDERLIMIT>
    </prefix1:CUSTOMER_ORDER>
  </prefix1:CUSTOMER>
</prefix1:CustOrderStatus>
```

3. Return to Design mode.

Ex 7: Step 8. Determine the Amount of Currently Open Orders

Follow these steps to find the total amount of open orders in the sample PB-BB database:

1. Click the XQuery Functions.
2. Drag and drop the multiply [`*`] operator into the design area.
3. Drag and drop `[PB-BB]/db/CUSTOMER_ORDER_LINE_ITEM/QUANTITY` and `[PB-BB]/db/CUSTOMER_ORDER_LINE_ITEM/PRICE` to the function operands.
4. Drag and drop another `xf:sum` into the design area. Drag the output result of the `f()*2 - Function` (`*2` being the second use of a multiplication function) to the input parameter of `xf:sum2`.
5. Drag and drop output from `xf:sum2` onto the target schema `[COCPTSampleTargetSchema.xsd]/cocpt:CustOrderStatus/CUSTOMER/CUSTOMER_ORDER/OPEN_ORDER_TOTAL_AMOUNT`

Finally, we need to restrict results to open orders:

1. In the Data View Builder select Design —> Toolbox.
2. Click on Constants.

3. Create a string constant called `OPEN` and drag the icon to the right of `OPEN` to the `[PB-BB]/db/CUSTOMER_ORDER_LINE_ITEM/STATUS` item.

   If we run the query, the amount of open orders should be `150000`.

Ex 7: Step 9: Determine the Total Amount of All Open and New Orders

1. Click XQuery Functions.
2. Drag and drop the plus [+] numeric operator into the design area.
3. Use `xf:sum` (total new orders) and `xf:sum2` (total open orders) as the operands to obtain the total of open and new order amount.

Ex 7: Step 10: Test If Open Orders + New Orders Exceeds the Order Limit

1. Select XQuery Functions.
2. Under Comparison operators locate the [`gt`] (greater than) function. Drag it into the design area.
3. Use the output of the sum of open and newly arrived orders as the first input parameter.
4. Use the query parameter `orderLimit` as input for `operand2`. The result is a Boolean value that is True if the sum of open and newly arrived orders is greater than the overall order limit.

Ex 7: Step 11: Determine If the Order is Accepted or Rejected

Now that the relationships and conditions are established, set up an if-then-else test to solve the business problem. See "The Problem" on page 2-45.

1. Under Other functions locate the `xfext:if-then-else` function. Drag it into the design area.
2. Drag the output of the `gt` function to the `if-then-else` condition parameter.
3. Click on the Toolbox `Constants` tab. In the String field enter `REJECT`, drag the String field icon to the `then` parameter in the `xfext:if-then-else` function.
4. Change the String field to `ACCEPT`, then drag the String field icon to the `else` parameter.
5. Drag the output result onto the target schema `[COCPTSampleTargetSchema.xsd]/cocpt:CustOrderStatus/CUSTOMER/CUSTOMER_ORDER/ORDER_REVIEW_STATUS`
6. Click Test.

Ex 7: Step 12: View the XQuery

The generated XQuery is shown in the following code listing.

Listing 2-22 XQuery for Example 6: Complex Parameter Type (CPT)

```
namespace cocpt = "urn:schemas-bea-com:ld-cocpt"

<cocpt:CustOrderStatus>
   {
   for $COCPTSAMPLE.CUSTOMER_ORDER_2 in ($#COCPTSAMPLE of type element
cocpt:CustOrder)/CUSTOMER_ORDER
   for $PB_BB.CUSTOMER_3 in document("PB-BB")/db/CUSTOMER
   let $srcval_4 :=
                  for $PB_BB.CUSTOMER_ORDER_LINE_ITEM_5 in
document("PB-BB")/db/CUSTOMER_ORDER_LINE_ITEM
                  where xf:not(xf:empty(
                  for $PB_BB.CUSTOMER_ORDER_7 in
document("PB-BB")/db/CUSTOMER_ORDER
                  where ($PB_BB.CUSTOMER_3/CUSTOMER_ID  eq
$PB_BB.CUSTOMER_ORDER_7/CUSTOMER_ID)
                    and ($PB_BB.CUSTOMER_ORDER_7/ORDER_ID  eq
$PB_BB.CUSTOMER_ORDER_LINE_ITEM_5/ORDER_ID)
                  return
                  xf:true()))
                    and ("OPEN"  eq  $PB_BB.CUSTOMER_ORDER_LINE_ITEM_5/STATUS)
                  return
                  $PB_BB.CUSTOMER_ORDER_LINE_ITEM_5/QUANTITY *
$PB_BB.CUSTOMER_ORDER_LINE_ITEM_5/PRICE
   let $xf:sum2_8 := xf:sum($srcval_4)
   let $srcval_9 :=
                  for $COCPTSAMPLE.NEW_ORDER_LINE_ITEM_10 in
$COCPTSAMPLE.CUSTOMER_ORDER_2/NEW_ORDER_LINE_ITEM
                  return
                  $COCPTSAMPLE.NEW_ORDER_LINE_ITEM_10/QUANTITY *
$COCPTSAMPLE.NEW_ORDER_LINE_ITEM_10/PRICE
   let $xf:sum_12 := xf:sum($srcval_9)
   let $v_13 := $xf:sum2_8 + $xf:sum_12
   let $gt_14 := $v_13 gt $#orderLimit of type xs:decimal
   where ($COCPTSAMPLE.CUSTOMER_ORDER_2/CUSTOMER_ID  eq
$PB_BB.CUSTOMER_3/CUSTOMER_ID)
   return
   <cocpt:CUSTOMER>
      <cocpt:FIRST_NAME>{ xf:data($PB_BB.CUSTOMER_3/FIRST_NAME)
}</cocpt:FIRST_NAME>
      <cocpt:LAST_NAME>{ xf:data($PB_BB.CUSTOMER_3/LAST_NAME)
}</cocpt:LAST_NAME>
```

```
      <cocpt:CUSTOMER_ID>{ xf:data($COCPTSAMPLE.CUSTOMER_ORDER_2/CUSTOMER_ID)
}</cocpt:CUSTOMER_ID>
      <cocpt:CUSTOMER_ORDER>
         <cocpt:OPENORDER_TOTAL_AMOUNT>{ $xf:sum2_8
}</cocpt:OPENORDER_TOTAL_AMOUNT>
         <cocpt:NEWORDER_TOTAL_AMOUNT>{ $xf:sum_12
}</cocpt:NEWORDER_TOTAL_AMOUNT>
         <cocpt:TOTAL_OPEN_ORDERLIMIT>{ $#orderLimit of type xs:decimal
}</cocpt:TOTAL_OPEN_ORDERLIMIT>
         <cocpt:ORDER_REVIEW_STATUS>{ xfext:if-then-else( treat as
xs:boolean($gt_14), "REJECT", "ACCEPT") }</cocpt:ORDER_REVIEW_STATUS>
      </cocpt:CUSTOMER_ORDER>
   </cocpt:CUSTOMER>
   }
</cocpt:CustOrderStatus>
```

7. Return to Design mode.

Ex 7: Step 13. Run the XQuery to Verify the Result

When you run this query on the sample data sources as described in this example, the result is an accepted order.

Listing 2-23 Result of Example 6: Complex Parameter Type (CPT)

```
<prefix1:CustOrderStatus xmlns:prefix1="urn:schemas-bea-com:ld-cocpt">
  <prefix1:CUSTOMER>
    <prefix1:FIRST_NAME>JOHN_B_1</prefix1:FIRST_NAME>
    <prefix1:LAST_NAME>KAY_1</prefix1:LAST_NAME>
    <prefix1:CUSTOMER_ID>CUSTOMER_1</prefix1:CUSTOMER_ID>
    <prefix1:CUSTOMER_ORDER>
      <prefix1:OPENORDER_TOTAL_AMOUNT>150000</prefix1:OPENORDER_TOTAL_AMOUNT>
      <prefix1:NEWORDER_TOTAL_AMOUNT>40000</prefix1:NEWORDER_TOTAL_AMOUNT>
      <prefix1:TOTAL_OPEN_ORDERLIMIT>200000</prefix1:TOTAL_OPEN_ORDERLIMIT>
      <prefix1:ORDER_REVIEW_STATUS>ACCEPT</prefix1:ORDER_REVIEW_STATUS>
    </prefix1:CUSTOMER_ORDER>
  </prefix1:CUSTOMER>
</prefix1:CustOrderStatus>
```

CHAPTER 3

Samples Installed with Liquid Data

This chapter describes samples installed with Liquid Data which are arranged in the following sections:

- Simple Liquid Data Queries
- Complex Parameter Type (CPT) Sample Queries
- Data View Sample Queries
- Application View Sample Queries
- Miscellaneous Samples
  - Custom Functions (DB-UDF) Sample Query
  - DB-Web Service Sample Query
  - SQL_Call Sample Query
  - EJB API Sample

Sample queries can be found at:

```
<WL_HOME>/samples/liquiddata/
```

In the following sub-directories:

- `buildQuery` contains all sample queries other than EJB programming examples and the retail sample application
- `ejbAPI` provides an EJB programming example

- `SampleApp` contains the Retail Sample Application, described in Chapter 1, "Understanding the Avitek Customer Self-Service Sample Application."

Simple Liquid Data Queries

This section describes several types of simple queries you can run and/or recreate in the Data View Builder.

DB-XML Sample Query

This section includes the following topics related to the DB-XML sample query:

What This Query Demonstrates

The sample order query demonstrates how to use Liquid Data to create an integrated view that shows the connection of two different Enterprise Information Systems (EIS), a database, and XML. Creating an integrated view provides the ability to seamlessly access separate EISs using a single query.

Business Scenario

A wireless service provider uses a relational database system to manage its customer and order information. Recently it purchased a broadband service from another company. The broadband customer and order information is managed by a non-DBMS system, e.g., XML file system. To provide an integrated customer service system, we use the Liquid Data engine to seamlessly access customer order information across different types of Enterprise Information Systems (EIS).

How to Run the Query

1. Start the Liquid Data Samples server.
2. Start the Data View Builder.
3. In the Data View Builder, open the following project file:

   ```
   <WL_HOME>/<LD_HOME>/liquiddata/buildquery/db-xml/e2e-order.qpr
   ```

4. Click the Test tab. (This shows the generated query statement.)
5. Click the Run Query button and view the results. Results include:
   - Last name: `KAY_1`
   - Two wireless orders: `1000` and `2000`
   - Two BroadBand orders: `1000` and `1500`

If You Want to Recreate the Query ...

You can find a detailed tutorial describing how to create a similar query in the Liquid Data *Getting Started* guide.

References

You can find the Wireless database schema at:

`<WL_HOME>/samples/domains/liquiddata/ldrepository/schemas/WIRELESSP.sql`

You can find the BroadBand XML schema at:

`<WL_HOME>/samples/domains/liquiddata/ldrepository/schemas/b-co.xsd`

You can find the target schema at:

`<WL_HOME>/samples/domains/liquiddata/ldrepository/schemas/customerOrderReport.xsd`

You can find the query statement at:

`<WL_HOME>/samples/domains/liquiddata/ldrepository/stored_queries/order.xq`

Data Transformation Sample Query

This section includes the following topics related to the Data Transformation Sample query:

What This Query Demonstrates

This query demonstrates the use of functions for:

- Simple data transformations such as to_lower_case, to_upper_case and concatenation
- Aggregation

Business Scenario

A telecom service provider has information on its BroadBand customers stored in one database system, represented in Liquid Data by a schema called BroadBand. Information about its wireless customers is in a different database system and is represented by a schema called Wireless.

The problem to be solved is that the customer representative needs to find out the average spending for customers who use both Broadband and Wireless services.

How to Run the Query

1. Start the Liquid Data Samples server.
2. Start the Data View Builder.

In the Data View Builder, open the following project file:

```
<WL_HOME>/<LD_HOME>/liquiddata/buildquery/dataTransform/data.qpr
```

3. Click the Test tab. (This shows the generated query statement.)
4. Click the Run Query button and view the result. For `CUSTOMER_1` results are:
   - `john_b_1`
   - `KAY_1`
   - `1375`

- KAY_1, JOHN_B_1

If You Want to Recreate the Query ...

You can use existing sample data sources when recreating this query.

Build the Query in the Data View Builder

There are several ways to effect data transformations. The approach taken in this example is to transform a data element to the point where it is ready to be mapped to the target schema and then do the mapping.

All the data transformation functions shown in this sample are available under in the XQuery Functions section of the Toolbox. The term *srcval* refer to the input side of an XQuery function.

There are six parts to the data transformation sample:

- Setting Up the Project
- Converting a String to Lower Case
- Converting a String to Upper Case
- Combining Two Strings (Concatenation)
- Determining Average Sale Price

Setting Up the Project

1. Create a new project.
2. Move the following data sources into the work area:

   Relational Databases:

   - PB-BB (BroadBand orders RDBMS)
   - PB-WL (Wireless orders RDBMS)
3. Set the target schema to `dta.xsd`
4. Click on the Toolbox tab.
5. Select Query Parameters.
6. In the name field enter `custID`.
7. Assign a type of xs:string. Then click Create.

8. Create a join (eq) between your new query parameter and the CUSTOMER_ID field in the BroadBand data source [PB-BB]:.

Join Element	Join Element
[Query Parameter] **custID**	[PB-BB]/db/CUSTOMER/**CUSTOMER_ID**

9. Create a join (eq) between the following pair of elements by dragging one element over the other:

Join Element	Join Element
[PB-BB]/db/CUSTOMER/**CUSTOMER_ID**	[PB-BB]/db/CUSTOMER_ORDER/**CUSTOMER_ID**
[PB-BB]/db/CUSTOMER/**CUSTOMER_ID**	[PB-WL]/db/CUSTOMER/**CUSTOMER_ID**
[PB-WL]/db/CUSTOMER/**CUSTOMER_ID**	[PB-WL]/db/CUSTOMER_ORDER/**CUSTOMER_ID**

Converting a String to Lower Case

1. Click XQuery Functions in the Toolbox.

1. Move the XQuery string function xf:lower-case into the work area.

2. Transform the Broadband [PB-BB] First Name element to lower case:

Source:[PB-BB]/db/	Target: String Function
CUSTOMER/**FIRST_NAME**	xf:lower-case/**srcval**

3. Map the output of the lower-case function to the target schema:

Source: String Function	Target: [dta.xsd]/results/
xf:lower-case/**result**	result/**FIRST_NAME**

4. Close xf:lower-case.

Converting a String to Upper Case

1. Move the XQuery string function xf:upper-case into the work area.
2. Transform the Broadband [PB-BB] Last Name element to upper case:

Source:[PB-BB]/db/	Target: String Function
CUSTOMER/**LAST_NAME**	xf:/upper-case/**srcval**

3. Map the output of the xf:upper-case function to the target schema:

Source: String Function	Target: [dta.xsd]/results/
xf:uppercase/**result**	result/**LAST_NAME**

4. Close xf:upper-case.

Combining Two Strings (Concatenation)

In this section you can create a string containing a concatenation of first and last name, last name first. This involves using the two XQuery concatenate functions you just moved into the work area. Once the string is built up, you can map it into the target schema.

1. Move two instances of the XQuery string concatenate function into the work area. They will be automatically labeled xf:concat and xf:concat2.
2. Click on the Constants button in the left pane. In the string field enter ", " (do not type the quotes, only the comma followed by a space).
3. Drag the BroadBand LAST_NAME element [PB-BB] to the first input field of the first concatenation function:

Source:[PB-BB]/db/	Target: String Function
CUSTOMER/**LAST_NAME**	[xf:concat]/**operand1**

4. Drag the string constant ", " to the second input field of the first concatenation function:

Source: Constant	Target: String Function
,	[xf:concat]/**operand2**

5. Populate the input fields of the second concatenation function with the output result of the first concatenation function and [PB-BB] FIRST_NAME.

Source: String Function	Target: String Function
[xf:concat]/**result**	[xf:concat]/**operand1**

.

Source:[PB-BB]/db/	Target: String Function
CUSTOMER/**FIRST_NAME**	[xf:concat]/**operand2**

6. Map the second concatenation function to the target schema:

Source: String Function	Target: [dta.xsd]/results/
[xf:concat]/**result**	result/**FULL_NAME**

7. Close both concatenation functions.

Determining Average Sale Price

You first need to get a total for both Wireless and BroadBand sales.

1. Move two instances of the XQuery aggregate sum function into the work area. They will be automatically labeled xf:sum and xf:sum2.

2. Move the BroadBand TOTAL_ORDER_AMOUNT element to the input field of the first sum and Wireless TOTAL_ORDER_AMOUNT to the input field of the second sum function:

Source:[PB-BB]/db/	Target: Aggregate Function
[PB-BB]/db/CUSTOMER_ORDER/**TOTAL_ORDER_AMOUNT**	[xf:sum]/**srcval** (BroadBand)
[PB-WL]/db/CUSTOMER_ORDER/**TOTAL_ORDER_AMOUNT**	[xf:sum]/**srcval** (Wireless)

3. Move an instance of the XQuery numeric plus (+) function into the work area.
4. Add together the TOTAL_ORDER_AMOUNT sums of Wireless and BroadBand:

Source: Aggregate Function	Target: Numeric Operator Function
[xf:sum]/**result** (Wireless)	[+]/**operand1**
[xf:sum]/**result** (BroadBand)	[+]/**operand2**

5. Move an instance of the XQuery numeric division (DIV) function into the work area.
6. Move the result of the addition of Wireless and BroadBand sales to the first DIV operand:

Source: Numeric Operator Function	Target: Numeric Operator Function
[+]/**result** (total order amount)	xf:div/**operand1**

7. Close instances of plus [xf:+] and sum [xf:sum].

Next you want to get the total number of sales for Wireless and BroadBand combined.

1. Move two instances of the XQuery aggregate xf:count function into the work area. They will be automatically labeled xf:count and xf:count2.

2. Move the BroadBand ORDER_ID element to the input field of the first xf:count function and Wireless ORDER_ID to the input field of the second xf:count function.

Source:[PB-BB]/db/	Target: Aggregate Function
[PB-BB]/db/CUSTOMER_ORDER/**ORDER_ID**	[xf:count]/**srcval** (BroadBand)
[PB-WL]/db/CUSTOMER_ORDER/**ORDER_ID**	[xf:count]/**srcval** (Wireless)

3. Move an instance of the XQuery numeric plus (+) function into the work area.
4. Add together the ORDER_ID count totals of Wireless and BroadBand:

Source: Aggregate Function	Target: Numeric Operator Function
[xf:count]/**result** (Wireless)	[+]/**operand1**
[xf:count]/**result** (BroadBand)	[+]/**operand2**

5. Move the result of the addition of Wireless and BroadBand sales counts to the second DIV operand:

Source: Numeric Operator Function	Target: Numeric Operator Function
[+]/**result** (total order amount)	xf:div/**operand2**

Map the output of the XQuery division function to the target schema:

Source: String Function	Target: [dta.xsd]/results/
xf:div/**result**	result/**average**

6. Enter Test mode.
7. Supply CUSTOMER_1 for the custID parameter.
8. Run your query. Results are:

```
- john_b_1
```

- KAY_1
- 1375
- KAY_1, JOHN_B_1

References

You can find the wireless data base schema file at:

```
<WL_HOME>/samples/domains/liquiddata/scripts/ddl/WIRELESSP.sql
```

You can find the BroadBand XML schema at:

```
<WL_HOME>/samples/domains/liquiddata/scripts/ddl/BROADBANDP.sql
```

You can find the target schema at:

```
<WL_HOME>/samples/domains/liquiddata/ldrepository/schemas/dta.xsd
```

DB-DB Sample Query

This section includes the following related to the DB-DB sample query:

What This Query Demonstrates

This query demonstrates how to use Liquid Data to construct a query that integrates data from two distributed heterogeneous database sources.

Traditionally, an application developer needs to write a set of data access methods and some join coding to complete the information extraction and assembly. This example demonstrates how you can use a single XQuery statement to declaratively accomplish the same task in much cleaner way.

Business Scenario

A service provider has customer information stored in one database server. All the promotion plans are managed by another database server. This sample query illustrates how to use Liquid Data to extract qualified promotion information for a customer based on the state the customer lives in.

How to Run the Query

1. Start the Liquid Data Samples server.
2. Start the Data View Builder.
3. In the Data View Builder, open the following project file:

   `<WL_HOME>/<LD_HOME>/liquiddata/buildquery/db-db/e2e-promotion.qpr`
4. Click the Test tab. (This shows the generated query statement.)
5. Click the "Run Query" button and view the result.

Results should include:

- Last name: `KAY_1`
- State: `TX`

- Promotion name: `WIRELESS UPSELL`
- Price: `$49.99`

If You Want to Recreate the Query ...

You can use existing sample data sources when recreating this query.

Build the Query in the Data View Builder

1. Create a new project.
2. Move the following Relational Database data source schemas into the work area:
   - PB-WL (Wireless)
   - PB-CR (Relationship Management)
3. Set `promotionData.xsd` from the repository schema directory as the target schema and expand the schema.
4. Map the following elements from the Wireless customer data source (PB-WL) to the target schema:

Source: [PB-WL]/db/	Target: [promotionData.xsd]/PromotionInfo/Customer_promotion
CUSTOMER/**FIRST_NAME**	CUSTOMER/**FIRST_NAME**
CUSTOMER/**LAST_NAME**	CUSTOMER/**LAST_NAME**
CUSTOMER/**CUSTOMER_ID**	CUSTOMER/**CUSTOMER_ID**
CUSTOMER/**STATE**	CUSTOMER/**STATE**

5. Map the following elements from the customer relationship data source (PB-CR) to the target schema:

Source: [PB-CR]/db/	Target: [promotionData.xsd]/PromotionInfo/Customer_promotion
PROMOTION/**PROMOTION_NAME**	PROMOTION_PLAN/**PROMOTION_NAME**
PROMOTION_PLAN/**PLAN_NAME**	PROMOTION_PLAN/**PLAN_NAME**

Source: [PB-CR]/db/	Target: [promotionData.xsd]/PromotionInfo/Customer_promotion
PROMOTION_PLAN/**FROM_DATE**	PROMOTION_PLAN/**FROM_DATE**
PROMOTION_PLAN/**TO_DATE**	PROMOTION_PLAN/**TO_DATE**
PROMOTION_PLAN/**PRICE**	PROMOTION_PLAN/**PRICE**

6. In the Toolbox, click on the Constants button, enter `CUSTOMER_1` in the string field.
7. Create equal joins (eq) between the following pairs of elements by dragging one element over the other:

Join Element	Join Element
[PB-CR]/db/PROMOTION/**PROMOTION_NAME**	[PB-CR]/db/PROMOTION_PLAN/**PROMOTION_NAME**
[PB-CR]/db/PROMOTION/**STATE**	[PB-WL]/db/CUSTOMER/**STATE**

8. Map the `CUSTOMER_1` constant to the Wireless data source `CUSTOMER_ID` element.

Source: Constant	Target: [PB-WL]/db/
CUSTOMER_1	CUSTOMER/**CUSTOMER_ID**

9. Enter Test mode and run your query.

Results should include:

- Last name: `KAY_1`
- State: `TX`
- Promotion name: `WIRELESS UPSELL`
- Price: `$49.99`

References

You can find the Wireless data base schema file at:

```
<WL_HOME>/samples/domains/liquiddata/scripts/ddl/WIRELESSP.sql
```

You can find the CRM database schema at:

```
<WL_HOME>/samples/domains/liquiddata/scripts/ddl/CRMP.sql
```

You can find the target schema at:

```
<WL_HOME>/samples/domains/liquiddata/ldrepository/schemas/promotionData.xsd
```

You can find the query statement at:

```
<WL_HOME>/samples/domains/liquiddata/ldrepository/stored_queries/promotion.xq
```

Complex Parameter Type (CPT) Sample Queries

DB-CPT Sample Query

This section includes the following topics related to the DB-CPT sample query:

What This Query Demonstrates

This example demonstrates use of Liquid Data to create a single query spanning two Enterprise Information Systems (EIS), a database, and a complex parameter type (CPT).

Business Scenario

A CRM service provider uses a relational database system to manage its promotion plan. A CRM CPT has the promotion plan name for a given state and wishes to extract the details of one or more matching plan name from the database. We use the Liquid Data engine to seamlessly access CRM information across different types of Enterprise Information Systems (EIS).

How to Run the Query

1. Start the Liquid Data Samples server.
2. Start the Data View Builder.
3. In the Data View Builder, open the following project file:

   ```
   <WL_HOME>/<LD_HOME>/liquiddata/buildquery/db-cpt/cptSample.qpr
   ```

4. Click the Test tab. (This shows the generated query statement.)
5. Specify the location of the CPT sample XML stream for the sample parameter:

   ```
   <WL_HOME>/samples/domains/liquiddata/ldrepository/xml_files/crm-p-cptSa
   mple.xml
   ```

6. Click the Run Query button and view the result. Five promotion plans (CA, TX, WA, AZ, NV) are returned.

If You Want to Create a Query that Uses a Complex Parameter Type (CPT)

You can find a detailed example of creating a query that uses CPTs in "Example 7: Complex Parameter Type (CPT)" on page 2-45.

References

You can find the CRM database schema at:

```
<WL_HOME>/samples/domains/liquiddata/ldrepository/schemas/CRMP.sql
```

You can find the CPTSAMPLE XML schema at:

```
<WL_HOME>/samples/domains/liquiddata/ldrepository/schemas/crm-p-cptSample.
xsd
```

You can find the target schema at:

```
<WL_HOME>/samples/domains/liquiddata/ldrepository/schemas/cpt_sample.xsd
```

You can find the query statement at:

```
<WL_HOME>/samples/domains/liquiddata/ldrepository/stored_queries/crm_cptSa
mple.xq
```

DB-CPTCO Sample Query

This section includes the following topics related to the DB-CPTCO sample query:

What This Query Demonstrates

This sample order query demonstrates how to use Liquid Data to create an integrated view that shows the connection of two different Enterprise Information (EIS) Systems, a database, and a complex parameter type (CPT).

Business Scenario

A BroadBand service provider uses a relational database system to manage its customer and order information. It received a new order for a given customer via XML mapped to a complex parameter type (CPT). The XML for CPT consists of a customer id along with one or more new orders specifying the price and quantity that the customer is ordering.

The objective is to accept or reject orders based on determining the total outstanding balance once the existing outstanding (unpaid) balance and the total value of the new order are added together.

If the result is above the limit, then the routine outputs the statement `Order Rejected`. Otherwise the statement `Order Accepted` is output.

It is noteworthy that these results can be obtained only after the Liquid Data engine accesses customer order information through separate EISs using a single query.

How to Run the Query

1. Start the Liquid Data Samples server.
2. Start the Data View Builder.
3. In the Data View Builder, open the following project file:

```
<WL_HOME>/<LD_HOME>/liquiddata/buildquery/db-cptco/coCPTSample.qpr
```

4. Click the Test tab. (This shows the generated query statement.)

5. Specify the location of the CPT sample XML stream. Navigate to:

```
<WL_HOME>/samples/domains/liquiddata/ldrepository/xml_files/coCptSample2.xml
```

6. Click the Run Query button to view the result. Results include:
   - Last name: `KAY_1`
   - Open orders: `150000`
   - New order amount: `40000`
   - Status: `Order Accepted`

If You Want to Create a Query That Use a Complex Parameter Type

You can find a detailed example of creating a query that uses CPTs in "Example 7: Complex Parameter Type (CPT)" on page 2-45.

References

You can find the CRM database schema at:

```
<WL_HOME>/samples/domains/liquiddata/ldrepository/schemas/CRMP.sql
```

You can find the CPTSAMPLE XML schema at:

```
<WL_HOME>/samples/domains/liquiddata/ldrepository/schemas/crm-p-cptSample.xsd
```

You can find the target schema at:

```
<WL_HOME>/samples/domains/liquiddata/ldrepository/schemas/cpt_sample.xsd
```

You can find the query statement at:

```
<WL_HOME>/samples/domains/liquiddata/ldrepository/stored_queries/crm_cptSample.xq
```

Data View Sample Queries

Simple Data View Sample Query

This section includes the following topics related to the simple data view sample query:

What This Query Demonstrates

This query demonstrates how to add data views (created with the Data View Builder) to the Liquid Data Server repository as data sources. Once configured (as shown in the following example), Data Views become data sources for any Data View Builder client that connects to the server.

Business Scenario

It's getting close to the end of the quarter. The Accounts Receivable department wants to collect on receivables. An “outstanding order” query has been generated by the MIS team in the Sales Department. It provides a listing of all the outstanding orders with unpaid balance.

However, only the order id is included in the list. The MIS team in the Account Receivable department will turn this query into a Data View and design a light-weight ad hoc query to find the name and address of the outstanding accounts with more than $1,000 in amount based on the order ID reported in the Data View source.

How To Run the Query

1. Start the Liquid Data Samples server.
2. Start the Data View Builder.
3. In the Data View Builder, open the following project file:

   ```
   <WL_HOME>/<LD_HOME>/liquiddata/buildquery/view/viewSample.qpr
   ```

4. Click the Test tab. (This shows the generated query statement.)
5. Click the run query button and view the result. Results include:

- Broadband order amount: `1500`
- Wireless order amount: `2000`

If You Want to Recreate the Query ...

You can use existing sample data sources when recreating this query.

Build the Query in the Data View Builder

1. Create a new project.
2. Move the following data sources into the work area:

   Relational Databases:

   - PB-BB (broadband orders RDBMS)
   - PB-WL (wireless orders RDBMS)

   Data Views:

   - V_SRC
3. Set the target schema to `viewtarget.xsd`
4. Map the following elements from the Wireless customer data source (PB-WL) to the target schema:

Source: [PB-WL]/db/	Target: [viewtarget.xsd]/ViewResult/
CUSTOMER/**FIRST_NAME**	wireless/**FIRST_NAME**
CUSTOMER/**LAST_NAME**	wireless/**LAST_NAME**
CUSTOMER/**CUSTOMER_ID**	wireless/**CUSTOMER_ID**
CUSTOMER_ORDER/**TOTAL_ORDER_AMOUNT**	wireless/**TOTAL_ORDER_AMOUNT**

5. Map the following elements from the BroadBand customer data source (PB-BB) to the target schema:

Source: [PB-BB]/db/	Target: [viewtarget.xsd]/ViewResult/
CUSTOMER/**FIRST_NAME**	broadband/**FIRST_NAME**
CUSTOMER/**LAST_NAME**	broadband/**LAST_NAME**
CUSTOMER/**CUSTOMER_ID**	broadband/**CUSTOMER_ID**
CUSTOMER_ORDER/**TOTAL_ORDER_AMOUNT**	broadband/**TOTAL_ORDER_AMOUNT**

6. Map the following elements from the data view orders data source (V_SRC) to the target schema:

Source: [V_SRC]/results/result/	Target: [viewtarget.xsd]/ViewResult/
broadband/order/**ORDER_ID**	broadband/**ORDER_ID**
wireless/order/**ORDER_ID**	wireless/**ORDER_ID**

7. Create a join (eq) between the following pairs of elements by dragging one element over the other:

Join Element	Join Element
[PB-BB]/db/CUSTOMER/**CUSTOMER_ID**	[PB-BB]/db/CUSTOMER/CUSTOMER_ORDER/**CUSTOMER_ID**
[PB-WL]/db/CUSTOMER/**CUSTOMER_ID**	[PB-WL]/db/CUSTOMER/CUSTOMER_ORDER/**CUSTOMER_ID**
[V_SRC]/results/result/broadband/order/**ORDER_ID**	[PB-BB]/db/CUSTOMER_ORDER/**ORDER_ID**
[V_SRC]/results/result/wireless/order/**ORDER_ID**	[PB-WL]/db/CUSTOMER_ORDER/**ORDER_ID**

8. In the Toolbox choose the greater than function (xf:gt) from Comparison Operators.

9. Click on XQuery Functions. Under Comparison Operators locate the gt function. Move two instances into the work area. These will automatically be labeled xf:gt and xf:gt2.
10. Click on the Constants button in the left pane, enter `1000` in the number field. Then click Ok.
11. Create a comparison that determines whether BroadBand (PB-BB) CUSTOMER_ORDER/TOTAL_ORDER_AMOUNT is greater than (gt) 1,000. Then associate the GT test result with TOTAL_ORDER_AMOUNT.

Source	Target: Comparison Function
[PB-BB]/db/CUSTOMER_ORDER/**TOTAL_ORDER_AMOUNT**	[GT]/**anyValue1**
[CONSTANT]/**1000**	[GT]/**anyValue2**
[GT]/**result**	[PB-BB]/db/CUSTOMER_ORDER/**TOTAL_ORDER_AMOUNT**

12. Close the GT function whose results you just mapped.
13. Using the second GT function repeat Step 11 using the Wireless (PB-WL) CUSTOMER_ORDER/TOTAL_ORDER_AMOUNT field and the 1000 constant.
14. Enter Test mode and run your query. Results include:
    - Broadband order amount: `1500`
    - Wireless order amount: `2000`

References

You can find the Wireless database schema file at:

```
<WL_HOME>/samples/domains/liquiddata/scripts/ddl/WIRELESSP.sql
```

You can find the BroadBand database schema at:

```
<WL_HOME>/samples/domains/liquiddata/scripts/ddl/BROADBANDP.sql
```

You can find the target schema at:

```
<WL_HOME>/samples/domains/liquiddata/ldrepository/schemas/viewtarget.xsd
```

You can find the query statement at:

```
<WL_HOME>/samples/domains/liquiddata/ldrepository/stored_queries/viewSa
mple.xq
```

Parameterized Data View Sample Queries

This section includes two parameterized data view sample queries.

pviewSample

The following topics related to the pviewSample parameterized data view sample query:

What This Queries Demonstrates

This parameterized sample query demonstrate how to construct and use a parameterized view.

Orders are aggregated from two different data sources: an EIS system (via J2EE connector architecture and a WebLogic Application View), and a relational data source.

The order view is constructed as a parameterized view with CUSTOMER_ID being the input parameter. Once the data view is constructed and configured, it can be used to construct other queries; for example, you can create a query that returns customer information and all detail order information from the parameterized view.

How to Run the Query

1. Start the Liquid Data Samples server.
2. Start the Data View Builder.
3. In the Data View Builder, open either of the following project files:

```
<WL_HOME>/<LD_HOME>/liquiddata/buildquery/parameterized-view/pviewSample.qpr
```

4. Click the Test tab. (This shows the generated query statement.)
5. Click the Run Query button and view the result for the `CUSTOMER_2` customer ID. Results include:
   - Customer ID: `CUSTOMER_2`
   - Last name: `KAY_2`

- Three BroadBand orders: `1000`, `1500`, `2000`
- Three Wireless orders: `1000`, `2000`, `4000`

If You Want to Create a Query That Uses Parameterized Views

You can find a detailed example describing how to create a parameterized query in "Creating a Parameterized Data View" in *Building Queries and Data Views*.

References

You can find the sample Wireless database schema file at:

```
<WL_HOME>/samples/domains/liquiddata/scripts/ddl/WIRELESSP.sql
```

You can find the sample BroadBand database schema at:

```
<WL_HOME>/samples/domains/liquiddata/scripts/ddl/BROADBANDP.sql
```

You can find the sample target schema at:

```
<WL_HOME>/samples/domains/liquiddata/ldrepository/schemas/pviewsrc.xsd
```

and

```
<WL_HOME>/samples/domains/liquiddata/ldrepository/schemas/pviewsrc.xsd/pviewSa
mple.xsd
```

You can find the sample query statements at:

```
<WL_HOME>/samples/domains/liquiddata/ldrepository/stored_queries/pviewsrc.xq
```

and

```
<WL_HOME>/samples/domains/liquiddata/ldrepository/stored_queries/pviewSample.x
q
```

pviewSample1

The following topics related to the pviewSample1 parameterized data view sample query:

- What This Queries Demonstrates
- How to Run the Query
- If You Want to Create a Query That Uses Parameterized Views
- References

What This Query Demonstrates

This parameterized sample query demonstrate how to construct and use a parameterized view.

Orders are aggregated from two different data sources: a web service and a relational data source.

The order view is constructed as a parameterized view with CUSTOMER_ID being the input parameter. Once the data view is constructed and configured, it can be used to construct other queries; for example, you can create a query that returns customer information and all detail order information from the parameterized view.

How to Run the Query

1. Start the Liquid Data Samples server.
2. Start the Data View Builder.
3. In the Data View Builder, open either of the following project files:

```
<WL_HOME>/<LD_HOME>/liquiddata/buildquery/parameterized-view/pviewSample1.qpr
```

4. Click the Test tab. (This shows the generated query statement.)
5. Click the Run Query button and view the result for the `CUSTOMER_2` customer ID. Results include:
   - Customer ID: `CUSTOMER_2`
   - Last name: `KAY_2`
   - Three BroadBand orders: `1000`, `1500`, `2000`
   - Two Wireless orders: `1000`, `2000`

If You Want to Create a Query That Uses Parameterized Views

You can find a detailed example describing how to create a parameterized query in "Creating a Parameterized Data View" in *Building Queries and Data Views*.

References

You can find the sample Wireless database schema file at:

```
<WL_HOME>/samples/domains/liquiddata/scripts/ddl/WIRELESSP.sql
```

You can find the sample BroadBand database schema at:

```
<WL_HOME>/samples/domains/liquiddata/scripts/ddl/BROADBANDP.sql
```

You can find the target schema at:

```
<WL_HOME>/samples/domains/liquiddata/ldrepository/schemas/pviewsrc.xsd
```

and

```
<WL_HOME>/samples/domains/liquiddata/ldrepository/schemas/pviewsrc.xsd/pviewSa
mple.xsd
```

You can view the query statements at:

```
<WL_HOME>/samples/domains/liquiddata/ldrepository/stored_queries/pviewsrc1.xq
```

and

```
<WL_HOME>/samples/domains/liquiddata/ldrepository/stored_queries/pviewSample1.
xq
```

Application View Sample Queries

DB-AppView (Three Data Source) Sample Query

This section includes the following topics related to the DB-AppView sample query that uses three data sources:

What This Query Demonstrates

This sample illustrates how to use LiquidData to extract customer information from three heterogeneous data sources: a JCA/Application View, an XML file, and a relational database.

Business Scenario

Customer information is stored in XML; customer orders can only be accessed through an EIS system (via J2EE Connector Architecture and WebLogic Application View); and promotion information is stored in a relational database. Traditionally, an application developer needs to write a set of data access methods and some application logic to complete the information extraction and assemble the result. Using Liquid Data, a single XQuery statement is used to accomplish this task.

How to Run the Query

1. Start the Liquid Data Samples server.
2. Start the Data View Builder.
3. In the Data View Builder, open the following project file:

   `<WL_HOME>/<LD_HOME>/liquiddata/buildquery/db-appview-xml/threeds.qpr`
4. Click the Test tab. (This shows the generated query statement.)
5. Click the Run Query button to view the result. Results include:
   - First name: `JOHN_1`
   - Last name: `KAY_1`

- Two customer orders in the amounts of `1000` and `2000`
- One `$49.99` promotion plan called `Family Holiday Connect`

If You Want to Recreate the Query ...

You can use existing sample data sources when recreating this query.

Build the Query in the Data View Builder

1. Create a new project.
2. Move the following data sources into the work area:

   XML Files:

   - XM-WL-C (wireless customers)

   Relational Databases:

   - PB-CR (promotion plan RDBMS)

   Application Views:

   - AV-WL/getCustomerOrder (wireless customer orders)
3. Set the target schema to `threeds.xsd`
4. Map the following elements from the Wireless customer XML data source (XM-WL-C) to the target schema:

Source: [XM-WL-C]/db/	Target: [threeds.xsd]/CUSTOMERINFO/
CUSTOMER/**FIRST_NAME**	CUSTOMER/**FIRST_NAME**
CUSTOMER/**LAST_NAME**	CUSTOMER/**LAST_NAME**
CUSTOMER/**CUSTOMER_ID**	CUSTOMER/**CUSTOMER_ID**

5. Map the following elements from the Wireless orders application view (AV-WL) to the target schema:

Source: [AV-WL]/getCustomerOrder/	Target: [threeds.xsd]/CUSTOMERINFO/
ns:Rows/Row/**TOTAL_ORDER_AMOUNT**	ORDERS/ORDER/**TOTAL_ORDER_AMOUNT**
ns:Rows/Row/**ORDER_ID**	ORDERS/ORDER/**ORDER_ID**
ns:Rows/Row/**SHIP_METHOD**	ORDERS/ORDER/**SHIP_METHOD**

6. Map the following elements from the Relational promotion plan data source (PB-CR) to the target schema:

Source: [PB-CR]/db/	Target: [threeds.xsd]/CUSTOMERINFO/
PROMOTION/**PROMOTION_NAME**	PROMOTIONPLANS/PROMOTIONPLAN/**PROMOTION_NAME**
PROMOTION_PLAN/**FROM_DATE**	PROMOTIONPLANS/PROMOTIONPLAN/**FROM_DATE**
PROMOTION_PLAN/**TO_DATE**	PROMOTIONPLANS/PROMOTIONPLAN/**TO_DATE**
PROMOTION_PLAN/**PRICE**	PROMOTIONPLANS/PROMOTIONPLAN/**PRICE**

7. Click on the Toolbox tab.
8. Click Constants. Enter `CUSTOMER_1` in the String field.
9. Create joins (eq) between the following pairs of elements by dragging one element over the other:

Join Element	Join Element
[XM-WL-C]/db/CUSTOMER/**CUSTOMER_ID**	[AV-WL:getCustomerOrder]/ns1:Input/**CUSTOMER_ID**
[XM-WL-C]/db/CUSTOMER/**STATE**	[PB-CR]/db/PROMOTION/**STATE**
[PB-CR]/db/PROMOTION/**PROMOTION_NAME**	[PB-CR]/db/PROMOTION_PLAN/**PROMOTION_NAME**

10. Map the CUSTOMER_1 constant to the Wireless Customer [XM-WL-C] CUSTOMER_ID field:

Source: Constant	Target: [XM-WL-C]/db/
CUSTOMER_1	CUSTOMER/**CUSTOMER_ID**

11. Create the following greater than (GT) tests which will filter out orders less than $1,000:
12. Run your query. Results should include:

- First name: `JOHN_1`
- Last name: `KAY_1`
- Two customer orders in the amounts of `1000` and `2000`
- One `$49.99` promotion plan called `Family Holiday Connect`

Reference

You can find the wireless data base schema file at:

```
<WL_HOME>/samples/domains/liquiddata/scripts/ddl/WIRELESSP.sql
```

You can find the CRM database schema at:

```
<WL_HOME>/samples/domains/liquiddata/scripts/ddl/CRMP.sql
```

You can find the target schema at:

```
<WL_HOME>/samples/domains/liquiddata/ldrepository/schemas/threeds.xsd
```

DB-AppView (Two Data Source) Sample Query

This section includes the following topics related to the DB-AppVIew sample query that uses two data sources:

What This Query Demonstrates

The sample for customer care query illustrates how to use Liquid Data to extract customer order handling information from two data sources:

- a JCA Application View
- a relational database containing customer information

Business Scenario

Due to historical reasons a telecom company has two separate order management systems: one for wireless service and the other for broadband. The wireless customer order information can only be accessed as an EIS system (via J2EE Connector Architecture and WebLogic Application View), and the broadband customer order information can only be accessed through its relational database.

The customer ID is the same across two systems. Traditionally, an application developer needs to write a set of data access methods and some application logic to complete the information extraction and assembly. Using Liquid Data, we demonstrate the use of one single Xquery statement to declaratively accomplish the same task.

How to Run the Query

1. Start the Liquid Data Samples server.
2. Start the Data View Builder.
3. In the Data View Builder, open the following project file:

   ```
   <WL_HOME>/<LD_HOME>/liquiddata/buildquery/db-appview/db_appview.qpr
   ```

4. Click the Test tab. (This shows the generated query statement.)

5. Click the Run Query button and view the result. Results include:
   - First name: `JOHN_1`
   - Last name: `KAY_1`
   - Orders: Six orders (three for `200`; three for `300`)

If You Want to Recreate the Query ...

You can use existing sample data sources when recreating this query.

Build the Query in the Data View Builder

1. Create a new project.
2. Move the following data sources into the work area:

   Relational Databases:
   - PB-BB (broadband customer RDBMS)

   Application Views:
   - AV-WL/AV-WL/getCustomerByFullName (wireless customer orders)
3. Set the target schema to `customerOrders.xsd`
4. Click the Toolbox tab.
5. Click the Constants button, enter `JOHN_1` in the string field.
6. Create a join (eq) between the following elements by dragging one element over the other:

Source: Constant	Target: [AV-WL]:getCustomerByFullName/
JOHN_1	ns3:Input/**FIRST_NAME**

7. Enter `KAY_1` in the constant string field.

8. Create an equal join (eq) between the following elements by dragging one element over the other:

Source: Constant	Target: [AV-WL]:getCustomerByFullName/
KAY_1	ns3:Input/**LAST_NAME**

9. Map the following elements from the AV-WL data source to the target schema:

Source: [AV-WL]:getCustomerByFullName/	Target: [customerOrders.xsd]/customers/
ns2:Rows/Row/CUSTOMER_ID/**FIRST_NAME**	customer/**first_name**
ns2:Rows/Row/CUSTOMER_ID/**LAST_NAME**	customer/**last_name**
ns2:Rows/Row/CUSTOMER_ID/**CUSTOMER_ID**	customer/**id**

10. Create a join [eq] between the following elements by dragging one element over the other:

Join Element	Join Element
[AV-WL]/getCustomerByFullName/ns2:Rows/Row/**CUSTOMER_ID**	[PB-BB]/db/CUSTOMER/CUSTOMER_ORDER/**CUSTOMER_ID**
[PB-BB]/db/CUSTOMER/CUSTOMER_ORDER/**ORDER_ID**	[PB-BB]/db/CUSTOMER/CUSTOMER_ORDER_LINE_ITEM/**ORDER_ID**

11. Map the following elements from the BroadBand customer relation data source [PB-BB] to the target schema:

Source: [PB-BB]/db/CUSTOMER/	Target: [customerOrders.xsd]/customers/
CUSTOMER_ORDER/**ORDER_DATE**	orders/order/**date**
CUSTOMER_ORDER/**ORDER_ID**	orders/order/**id**
CUSTOMER_ORDER_LINE_ITEM/**PRICE**	orders/order/**amount**

12. In the Constants string field enter `OPEN`.
13. Map the OPEN constant to the BroadBand orders [PB-BB] CUSTOMER_ORDER_LINE_ITEM by dragging one element over the other:

Source: Constant	Target: [PB-BB]/db/CUSTOMER/
OPEN	CUSTOMER_ORDER_LINE_ITEM/**STATUS**

14. Enter Test mode and run your query. Results include:
    - First name: `JOHN_1`
    - Last name: `KAY_1`
    - Orders: Six orders (three for `200`; three for `300`)

References

You can find the BroadBand database schema file at:

```
<WL_HOME>/samples/domains/liquiddata/scripts/ddl/BROADBANDP.sql
```

You can find the target schema at:

```
<WL_HOME>/samples/domains/liquiddata/ldrepository/schemas/customerOrder
s.xsd
```

You can find the query statement at:

```
<WL_HOME>/samples/domains/liquiddata/ldrepository/stored_queries/db_app
view.xq
```

Miscellaneous Samples

This section contains several samples, including:

- Stored Procedure Sample Query
- Custom Functions (DB-UDF) Sample Query
- DB-Web Service Sample Query
- EJB API Sample

Stored Procedure Sample Query

This section includes the following topics related to the stored procedure sample query:

- What This Query Demonstrates
- How to Run the Query
- If You Want to Create a Query That Uses Stored Procedures
- References

What This Query Demonstrates

This query demonstrates how to use RDBMS stored procedures as Liquid Data data sources. Once configured a RDBMS stored procedure becomes available as a function to any Data View Builder client that connects to the Liquid Data Server.

Business Scenario

For example, the MIS Department of a wireless service provider uses a database management system to manage its customer and order information. The engineers have already developed a full set of stored procedures with embedded business logic.

Liquid Data allows them to treat a stored procedure as a regular function that users can build queries on top of it. All the existing investment in stored procedure will be preserved.

In this example, a stored procedure is developed to report the total outstanding balance and number of outstanding orders for a particular customer. The stored procedure is used in a joint query with another data source.

How to Run the Query

1. Start the Liquid Data Samples server.
2. Start the Data View Builder.
3. In the Data View Builder, open the following project file:

   `<WL_HOME>/<LD_HOME>/liquiddata/buildquery/stored-procedure/GetOrderInfo.qpr`

4. Click the Test tab. (This shows the generated query statement.)
5. Click the Run Query button and view the result. Results include:
   - Sum: `3000`
   - Totalorders: `2`

If You Want to Create a Query That Uses Stored Procedures

You can find a detailed example describing how to create a stored procedure in Defining Stored Procedures to Liquid Data in *Building Queries and Data Views*.

References

You can find the target schema at:

`<WL_HOME>/samples/domains/liquiddata/ldrepository/schemas/getorderinfo.xsd`

You can find the query statement at:

`<WL_HOME>/<LD_HOME>/liquiddata/buildquery/stored-procedure/getorderinfo.xq`

Custom Functions (DB-UDF) Sample Query

This section includes the following topics related to the DB-UDF sample query:

What this Query Demonstrates

This query demonstrates how to create a custom function (also known as a *user-defined* function). Once the custom functions are defined you can use them in XQuery statements as you would any built-in Liquid Data function.

For example, suppose you need to access legacy data via a custom interface such as a session bean, entity bean, stored procedure and so on. Assume the custom interface is the only way to get data that is needed for another query. In this example, the custom interface is exposed by a session bean which implements two functions: getCustomerOrder( ) and getCustomer( ).

By exposing the interface methods as custom functions, the need to expose the data source and hence all the data in it, to the Liquid Data engine is eliminated.

A custom function only delivers a subset of data from the data source (in this case, Customer Orders and Customers) without exposing other tables and content.

How to Run the Queries

1. Start the Liquid Data Samples server.
2. Start the Data View Builder.
3. In the Data View Builder, open either of the following project files:

```
<WL_HOME>/<LD_HOME>/liquiddata/buildquery/db-udf/src/sampleProjects/dvbProjects/CustUdf.qpr
```

```
<WL_HOME>/<LD_HOME>/liquiddata/buildquery/db-udf/src/sampleProjects/dvbProjects/CustOrdUdf.qpr
```

4. Click the Test tab. (This shows the generated query statement.)
5. Click the Run Query button and view the result.

For the CustUdf query results include:

- First name: `JOHN_B_1`
- Last name: `KAY_1`

For the CustOrdUdf query results include:

– Two orders, one for `1000` and one for `1500`.

If You Want to Recreate the Custom Functions and the Queries ...

You can use existing sample data sources when recreating this query.

Create a CFLD file

A CFLD file is where you define your function along with a schema associated with the function's return type. This sample uses the cfld file located at:

```
<WL_HOME>/<LD_HOME>/liquiddata/buildquery/db-udf/src/examples/ldi/userDefinedF
unc/UserDefinedFunction.cfld
```

The cfld is copied to the Repository under `custom_functions` folder.

Listing 3-1

```
<types>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">

<!-- The schema describing the CustomerOrder object returned -->

<xs:element name="CustomerOrder">
<xs:complexType>
<xs:sequence>
<xs:element ref="ORDER_DATE"/>
<xs:element ref="ORDER_ID"/>
<xs:element ref="CUSTOMER_ID"/>
<xs:element ref="SHIP_METHOD"/>
<xs:element ref="TOTAL_ORDER_AMOUNT"/>
</xs:sequence>
</xs:complexType>
</xs:element>
<xs:element name="ORDER_DATE" type="xs:date"/>
<xs:element name="ORDER_ID" type="xs:string"/>
<xs:element name="SHIP_METHOD" type="xs:string"/>
```

```
<xs:element name="TOTAL_ORDER_AMOUNT" type="xs:decimal"/>
<xs:element name="CUSTOMER_ID" type="xs:string"/>
<xs:element name="CustomerOrders">
<xs:complexType>
<xs:sequence>
<xs:element ref="CustomerOrder" minOccurs="0" maxOccurs="unbounded"/>
</xs:sequence>
</xs:complexType>
</xs:element>
</xs:schema>
</types>
```

Note that in the CFLD file shown above under TYPE we have defined a customerOrder Element consisting of ORDER_DATE,ORDER_ID,SHIP_METHOD,TOTAL_ORDER_AMOUNT,CUSTOMER_ID

Next create a function.

Listing 3-2

```
<functions>
<!-- name is the function name,
return_type : is the custom method return type
class: fully qualified classname
method: the method in the class to map to
argument: list of argument with type. This is are the input(s) to the method
with asscociated labels
-->
<!-- The function is mapping to getCustomerOrder in
examples.ldi.userDefinedFunc.UserFunctionMapping.class
It takes to input (url and customer_id), and returns a Element object of
type CustomerOrder
as defined above
-->
<function name="getCustomerOrder" return_type="CustomerOrders"
class="examples.ldi.userDefinedFunc.UserDefinedFuncMapping"
method="getCustomerOrder">
```

```
<argument type="xs:string" label="url"/>
<argument type="xs:string" label="customerID"/>
<presentation group="Sample custom functions" />
<description>Get Customer Order gets a list of customer order for a given
customer id </description>
</function>
```

Create a Mapping class

Finally, create a mapping class that maps the functions defined in the cfld with the actual implementation.

The mapping class used in this sample is located at:

```
<WL_HOME>/<LD_HOME>/liquiddata/buildquery/db-udf/src/examples/ldi/userDefinedF
unc/UserDefinedFuncMapping.java
```

```
public static Element getCustomerOrder(String url, String customerID)
```

The method getCustomerOrder( ) matches with the function description specified in the CFLD. Note that its return type is an Element (XML). But this element is described in the CFLD file as a type of customerOrders. Hence the function mapping is responsible for generating an element of type customerOrder. In our sample the mapping function does a JNDI lookup to call getCustomerOrder( ) in the session bean.

Build the getCustomer( ) Query in the Data View Builder

1. Create a new project.
2. Click on the Toolbox tab.
3. Move the following custom function (located in the Sample custom function folder) into the work area:
   - getCustomer( )
4. Set the target schema to `custUdf.xsd`

5. Map the following elements from the getCustomer( ) custom function to the target schema:

Source: [getCustomer]/Customers/	Target: [custUdf.xsd]/RESULT/
CUSTOMER/**FIRST_NAME**	Customer/**FIRST_NAME**
CUSTOMER/**LAST_NAME**	Customer/**LAST_NAME**
CUSTOMER/**CUSTOMER_ID**	Customer/**CUSTOMER_ID**

6. Click on the Constants button. Enter the URL `t3://localhost:7001` into the string field.
7. Map the URL address to the user-defined function url:

Source: Constant	Target:
/t3://localhost:7001	[UDF:getCustomer]/**url**

8. Change the string constant value to `CUSTOMER_1`.
9. Map the CUSTOMER_1 constant to the user-defined function Customer_ID element

Source: Constant	Target:
CUSTOMER_1	[UDF:getCustomer]/**CustomerID**

10. Click the Run Query button and view the result. Results include:
    - First name: `JOHN_B_1`
    - Last name: `KAY_1`
    - CustomerID: `CUSTOMER_1`

Build the getCustomerOrder( ) Query in the Data View Builder

1. Create a new project.
2. Click on the Toolbox tab.

3. Move an instance of the following custom function into the work area:
   – getCustomerOrder( )
4. Set the target schema to `custOrdUdf.xsd`
5. Map the following elements from the getCustomer( ) custom function to the target schema:

Source: [UDF:getCustomerOrder]/CustomerOrders/	**Target: [custOrdUdf.xsd]/RESULT/**
CustomerOrder/**ORDER_DATE**	CustomerOrder/**ORDER_DATE**
CustomerOrder/**ORDER_ID**	CustomerOrder/**ORDER_ID**
CustomerOrder/**CUSTOMER_ID**	CustomerOrder/**CUSTOMER_ID**
CustomerOrder/**SHIP_METHOD**	CustomerOrder/**SHIP_METHOD**
CustomerOrder/**TOTAL_ORDER_AMOUNT**	CustomerOrder/**TOTAL_ORDER_AMOUNT**

6. Click on the Constants button in the left pane, enter the URL `t3://localhost:7001` in the string field.
7. Map the URL to the appropriate input parameter of the user-defined function:

Source: Constant	**Target:**
/t3://localhost:7001	[UDF:getCustomerOrder]**/url**

8. Change the string constant value to `CUSTOMER_1`.
9. Map the string constant to the other input parameter of the user-defined function:

Source: Constant	**Target:**
CUSTOMER_1	[UDF:getCustomerOrder]/**CustomerID**

10. Enter Test mode and run the query. Results include:
    – Two orders, one for `1000` and one for `1500`.

If You Want to Build the Sample Source Code ...

If you want to build a custom function, you can build the sample source code as follows.

1. Run `setLDExampleEnv.cmd` or `setLDExampleEnv.sh` located at:

   `<WL_HOME>/samples/domains/liquiddata/`

   This will set up your environment variables needed to run the query.

2. Add `ant` to your path.

3. Compile the source code via `build.xml` using ant:

   ```
   cd <WL_HOME>/<LD_HOME>/liquiddata/buildquery/db-udf/build
   ant
   ```

   This will compile the script and generate the `.ear` and other `.jar` files for you in the output directory.

4. Find the compiled code at:

   `<WL_HOME>/<LD_HOME>/liquiddata/buildquery/db-udf/build/output.`

5. Copy the `ldsample_udf.ear` to the applications directory of your development domain:

```
<WL_HOME>/<LD_HOME>/liquiddata/buildquery/db-udf/build/output/applications/lds
ample_udf.ear
```

   to:

   `<WL_HOME>/samples/domains/liquiddata/applications`

6. Copy `ldsample_udf_rep.jar` and `ldsample_clientAPI.jar` to the domain's repository under the `custom_lib` folder.

7. Copy:

   `<WL_HOME>/<LD_HOME>/liquiddata/buildquery/db-udf/build/output/mapping_clas`
   `ses/*.jar`

   to

   `<WL_HOME>/samples/domains/liquiddata/ldrepository/custom_lib`

8. Copy `UserDefinedFuncMapping.cfld` to the repository under the `custom_functions` folder.

9. Copy:

```
<WL_HOME>/<LD_HOME>/liquiddata/buildquery/db-udf/src/examples/ldi/userDefi
nedFunc/UserDefinedFuncMapping.cfld
```

to:

```
<WL_HOME>/samples/domains/liquiddata/ldrepository/custom_functions
```

10. On the ldConsole server, Liquid Data node refresh or create a entry for functionResourcelib giving a name for the function group along with the name of the cfld file.

If you used Data View Builder, you will notice your custom functions have been added.

Reference

You can find the target schema that retrieves customers at:

```
<WL_HOME>/samples/domains/liquiddata/ldrepository/schemas/custUdf.xsd
```

You can find the target schema that retrieves customer orders at:

```
<WL_HOME>/samples/domains/liquiddata/ldrepository/schemas/custorderUdf.xsd
```

You can find the cfld file at:

```
<WL_HOME>/<LD_HOME>/liquiddata/buildquery/db-udf/src/examples/ldi/userDefinedF
unc/UserDefinedFunction.cfld
```

DB-Web Service Sample Query

This document includes the following topics related to the DB-Web Service sample query:

What This Query Demonstrates

This query demonstrates how to use Liquid Data to access separate Enterprise Information Systems (EIS) using a single query.

Business Scenario

A wireless service provider uses a relational database management system to manage its product information. The sales price of these product is computed using a Web Service. To provide an integrated product and price information, we use liquidData engine to seamlessly access product and price across EISs.

How to Run the Query

1. Start the Liquid Data Samples server.
2. Start the Data View Builder.
3. In the Data View Builder, open the following project file:

   `<WL_HOME>/<LD_HOME>/liquiddata/buildquery/db-webservice/WSPricer.qpr`
4. Click the Test tab. (This shows the generated query statement.)
5. Click the Run Query button and view the result. Results include:
   - Five products (`E110`, `E900`, `NOK9250`, `S6225`, `SS8`), each with a sales price of $100.

If You Want to Recreate the Query …

You can use existing sample data sources when recreating this query.

Build the Query in the Data View Builder

1. Create a new project.

2. Move the following data sources into the work area:

   Relational Databases:

   – PB-WL (wireless orders RDBMS)

   Web services:

   – Pricer:getSalesPrice( )

3. Set the target schema to `pricer.xsd`

4. Map the product price element from the Wireless product name to the input element of the getSalesPrice( ) function:

Source: [PB-WL]/db/	Target: Pricer:getSalesPrice/
PRODUCTS/**PRODUCT_NAME**	**string**

5. Map the product name from the Wireless order RDBMS data source (PB-WL) to the target schema:

Source: [PB-WL]/	Target: [pricer.xsd]/PRODUCTPRICE/
PRODUCTS/**PRODUCT_NAME**	**PRODUCT/NAME**

6. Map the product price from the Pricer:getSalesPrice( ) function to the target schema:

Source: Pricer:getSalesPrice/	Target: [pricer.xsd]/PRODUCTPRICE/
result	PRODUCT/**SALESPRICE**

7. Enter Test mode and run your query. Results include:

   – Five products (`E110`, `E900`, `NOK9250`, `S6225`, `SS8`), each with a sales price of $100.

References

You can find the wireless data base schema file at:

```
<WL_HOME>/samples/domains/liquiddata/scripts/ddl/WIRELESSP.sql
```

You can find the target schema at:

```
<WL_HOME>/samples/domains/liquiddata/ldrepository/schemas/pricer.xsd
```

You can access the WSDL File at:

```
http://localhost:7001/wspricer/Pricer?WSDL
```

You can find the query statement at:

```
<WL_HOME>/samples/domains/liquiddata/ldrepository/stored_queries/SalesP
rice.xq
```

SQL_Call Sample Query

This document includes the following topics related to the SQL_Call sample query:

What This Query Demonstrates

This query demonstrates how to use Liquid Data to construct a query that integrates data from a user-supplied SQL statement.

Business Scenario

For example, if there is a requirement to execute complicated or database-dependent SQL, the SQL Call mechanism can be used as a Liquid Data data source. (Note: the SQL Call mechanism should be used sparingly as it involves manual configuration and precludes certain optimizations that Liquid Data might otherwise provide.)

How to Run the Query

1. Start the Liquid Data Samples server.
2. Start the Data View Builder.
3. In the Data View Builder, open the following project file:

```
<WL_HOME>/samples/liquiddata/buildQuery/sql_call/GetOrderSummary.qpr
```

4. Click the Test tab. (This shows the generated query statement.)
5. Enter `CUSTOMER_%` for the parameter `CUSTOMER_ID_PATTERN`
6. Click the Run Query button and view the result. Results include:
   - California: 2 order count, average `1000`
   - Washington: 6 order count, average `2333.3333...`

If You Want to Recreate the Query ...

You can use existing sample data sources when recreating this query.

1. Create a new project.
2. Move the following data sources into the work area:

   SQL Calls:

   – PB-WL/GetOrderSummarySQL
3. From the File menu select Set selected schema as target schema.

   The elements in the PB-WL:GetOrderSummarySQL schema will be replicated in the target schema.
4. Click on the Toolbox tab and select Query Parameter.
5. Enter CUSTOMER_ID_PATTERN in the name field.
6. Select xs:string as the type and click Create.
7. Map the constant to the customer identifier:

Source: Constant	Target: [PB-WL:/GetOrderSummarySQL]/
CUSTOMER_ID_PATTERN	**CUSTOMER_ID_PATTERN**

8. Map the elements from the GetOrderSummarySQLCalls web service providing wireless order data source to the target schema:

Source: [PB-WL:GetOrderSummarySQL]/OrderSummarys/resultSetOrderSummary/	Target: [PB-WL:/GetOrderSummarySQL2]/OrderSummarys /resultSetOrderSummary/
orderSummaryRow/**STATE**	orderSummaryRow/**STATE**
orderSummaryRow/**ORDER_COUNT**	orderSummaryRow/**ORDER_COUNT**
orderSummaryRow/**AVERAGE**	orderSummaryRow/**AVERAGE**
orderSummaryRow/**MIN**	orderSummaryRow/**MIN**
orderSummaryRow/**MAX**	orderSummaryRow/**MAX**

9. Enter Test mode.

10. Enter `CUSTOMER_%` for the CUSTOMER_ID_PATTERN parameter.
11. Run the query. Results include:
    - California: 2 order count, average `1000`
    - Washington: 6 order count, average `2333.3333...`

SQL Call Description file

The SQL Call Description File (`pbsp.xsd`) defines a SQL Call named GetOrderSummary as the SQL shown below.

Listing 3-3 SQL Call Description File (pbsp.xsd)

```
SELECT
       state,
       count(STATE),
       avg(total_order_amount),
       min(total_order_amount),
       max(total_order_amount)
FROM
       customer,
       customer_order
WHERE
       ((state = 'CA') OR
        (state = 'OR') OR
        (state = 'WA')) AND
       customer.customer_id like ? AND
       customer.customer_id = customer_order.customer_id
GROUP BY state
```

References

- You can find the Wireless data base schema file at:

`<WL_HOME>/samples/domains/liquiddata/scripts/ddl/WIRELESSP.sql`

- You can find the target schema at:

`<WL_HOME>/samples/domains/liquiddata/ldrepository/schemas/getordersummary.xsd`

- You can find the SQL Call Definition File at:

```
<WL_HOME>/samples/domains/liquiddata/ldrepository/sql_calls/pbsp.xsd
```

- You can find the query statement at:

```
<WL_HOME>/samples/domains/liquiddata/ldrepository/stored_queries/getordersumma
ry.xq
```

CSV-XML Sample Query

This document includes the following topics related to the CSV-XML sample query:

What This Query Demonstrates

This sample customer information query demonstrates how to use Liquid Data to create an integrated view that shows the connection of two different data sources, a CSV (i.e. delimited) file which originated in a spreadsheet and an XML data source.

Creating an integrated view provides the ability to seamlessly access different sources using a single query.

Business Scenario

An XML file contains a larger set of information about customers (such as name, contact info, and so on) while a CSV file contains a smaller set of customers information such as age. The two sources share the same key: customerID. The goal of query is to gather all information about customers that are younger or at age 40.

How to Run the Query

1. Start the Liquid Data Samples server.
2. Start the Data View Builder.
3. In the Data View Builder, open the following project file:

```
<WL_HOME>/samples/liquiddata/buildQuery/csv-xml/GetCustomerByAge.qpr
```

4. Click the Test tab. (This shows the generated query statement.)
5. Click the Run Query button and view the result. Results include information on six customers all of whom are age 40 or younger.

If You Want to Recreate the Query ...

You can use existing sample data sources when recreating this query.

1. Create a new project.
2. Move the following data sources into the work area:

   XML:

   – XM-BB-C

   Delimited file:

   – cstest1

3. Create a join (eq) between the two CUSTOMER_ID source elements by dragging one element over the other:

Source: [xm-bb-c]/db/	Target: [cstest1]/CustomersInfo/
CUSTOMER/**CUSTOMER_ID**	CustomerInfo/**customer_ID**

4. Click on the XQuery Functions.
5. Open the Comparison operators folder.
6. Drag the less than [lt] function to the second line in the Conditions area.
7. Map the cstest1 CSV data source `age` element to the left side of the [lt] equation.
8. Click on the Toolbox tab and select Constants.
9. Enter `41` in the Number field.
10. Create a condition requiring that only customers age 40 or younger will be retrieved by the query through the following mappings:

Source: [cstest1]/CustomersInfo/	Target: [lt]/
CustomerInfo/**age**	**[left operand]**
Source: [CONSTANT]/	**Target: [lt]/**
41	**[right operand]**

11. Close the Function Editor.

12. In the XM-BB-C data source schema right-click on the CUSTOMER complex element name and choose Copy.
13. In the target schema right-click on `results`; choose Paste and Map.
14. Also in the target schema right-click on CUSTOMER and choose Expand complex mapping. (This is just an easier way of individually mapping the elements from the source to the target schema.)
15. From the File menu select Save Target Schema. Navigate to the Repository folder and save the schema file to the name `customerInfo2.xsd`.
16. In the `cstest1` source schema right-click on the `age` element and choose Copy.
17. In the target schema right-click on CUSTOMER and choose Paste. The `age` element appears in the target schema.
18. Map the `age` element from the CSV data source to the same name element in the target schema:

Source: [cstest1]/CustomersInfo/	Target: [customerInfo2]/results/
CustomerInfo/**age**	CUSTOMER/**age**

19. Click on Test mode and run the query. Results include information on six customers all of whom are age 40 or younger.

References

- You can find the CSV file schema at:

```
<WL_HOME>/samples/domains/liquiddata/ldrepository/schemas/csvtest1.xsd
```

- You can find the BroadBand XML schema at:

```
<WL_HOME>/samples/domains/liquiddata/ldrepository/schemas/b-c.xsd
```

- You can find the target schema at:

```
<WL_HOME>/samples/domains/liquiddata/ldrepository/schemas/getCustomerInfo.xsd
```

- You can find the query statement at:

```
<WL_HOME>/samples/domains/liquiddata/ldrepository/stored_queries/csv_xml.xq
```

EJB API Sample

This section references the EJB API sample query.

To build the EJBAPI testing classes

1. Navigate to the directory:

   ```
   <WL_HOME>/samples/domains/liquiddata/
   ```

   If you are on a Windows system run:

   ```
   setLDExamplesEnv.cmd
   ```

   Or if you are on a UNIX system run:

   ```
   . setLDExamplesEnv.sh
   ```

   on unix bsh)

---

2. go to:

   ```
   <WL_HOME>/samples/liquiddata/ejbAPI/build
   ```

3. run `ant` in the same directory. The class will be generated under

   ```
   <WL_HOME>/samples/liquiddata/ejbAPI/obj
   ```

To run the ejbAPI test classes

Make sure the sample server is started correctly and running in the non-secure mode.

1. Go to:

   ```
   <WL_HOME>/samples/liquiddata/
   ```

   run:

   `setLDExamplesEnv.cmd` (or run `setLDExamplesEnv.sh` on unix `bash`)

2. Go to:

   ```
   <WL_HOME>/samples/liquiddata/ejbAPI/obj
   ```

3. Enter the following string for:

   Windows:

   ```
   java -cp .;%CLASSPATH% ejbSample.QueryClient t3://localhost:7001
   ```

UNIX:

```
java -cp . :$CLASSPATH ejbSample.QueryClient t3://localhost:7001
```

You will see the result on the screen.

4. Enter the following string for:

   Windows:

```
java -cp .;%CLASSPATH% ejbSample.QueryParamClient t3://localhost:7001
orderparam CUSTOMER_1
```

   UNIX:

```
java -cp .:$CLASSPATH ejbSample.QueryParamClient t3://localhost:7001 orderparam
CUSTOMER_1
```

   you will see the result on the screen.

To examine the code

The source code is under the following directory:

```
<WL_HOME>/samples/liquiddata/ebjAPI/src
```

Index