

Oracle® WebLogic Integration

Transforming Data Using XQuery Mapper

10g Release 3 (10.3.1)

January 2010

ORACLE®

Oracle WebLogic Integration Transforming Data Using XQuery Mapper, 10g Release 3 (10.3.1)

Copyright © 2007, 2010, Oracle and/or its affiliates. All rights reserved.

This software and related documentation are provided under a license agreement containing restrictions on use and disclosure and are protected by intellectual property laws. Except as expressly permitted in your license agreement or allowed by law, you may not use, copy, reproduce, translate, broadcast, modify, license, transmit, distribute, exhibit, perform, publish, or display any part, in any form, or by any means. Reverse engineering, disassembly, or decompilation of this software, unless required by law for interoperability, is prohibited.

The information contained herein is subject to change without notice and is not warranted to be error-free. If you find any errors, please report them to us in writing.

If this software or related documentation is delivered to the U.S. Government or anyone licensing it on behalf of the U.S. Government, the following notice is applicable:

U.S. GOVERNMENT RIGHTS Programs, software, databases, and related documentation and technical data delivered to U.S. Government customers are "commercial computer software" or "commercial technical data" pursuant to the applicable Federal Acquisition Regulation and agency-specific supplemental regulations. As such, the use, duplication, disclosure, modification, and adaptation shall be subject to the restrictions and license terms set forth in the applicable Government contract, and, to the extent applicable by the terms of the Government contract, the additional rights set forth in FAR 52.227-19, Commercial Computer Software License (December 2007). Oracle USA, Inc., 500 Oracle Parkway, Redwood City, CA 94065.

This software is developed for general use in a variety of information management applications. It is not developed or intended for use in any inherently dangerous applications, including applications which may create a risk of personal injury. If you use this software in dangerous applications, then you shall be responsible to take all appropriate fail-safe, backup, redundancy, and other measures to ensure the safe use of this software. Oracle Corporation and its affiliates disclaim any liability for any damages caused by use of this software in dangerous applications.

Oracle is a registered trademark of Oracle Corporation and/or its affiliates. Other names may be trademarks of their respective owners.

This software and documentation may provide access to or information on content, products and services from third parties. Oracle Corporation and its affiliates are not responsible for and expressly disclaim all warranties of any kind with respect to third-party content, products, and services. Oracle Corporation and its affiliates will not be responsible for any loss, costs, or damages incurred due to your access to or use of third-party content, products, or services.

Contents

Introduction

Overview of XQuery Mapper	1-1
Support for XQuery 2002 and 2004.	1-2
Restrictions Applicable to the XQuery Test View	1-3

Transforming Data Using XQuery Mapper

Launching XQuery Mapper	2-2
Creating the XQuery Mapper Sample Project	2-2
Creating an XQuery Mapper Project	2-3
Importing and Creating Schema Files	2-4
Importing XML Schemas and MFL Files.	2-4
Creating XML Schemas	2-5
Creating XML Files from XML Schemas	2-6
Creating WSDL Files	2-7
Creating MFL Files.	2-8
Selecting Source and Target Data Types.	2-8
Creating Data Transformations	2-11
Creating Basic Element Transformations	2-11
Creating Basic Attribute Transformations	2-12
Creating Complex Transformations	2-13
Editing Data Transformations	2-14
Viewing and Editing XQuery Files.	2-14

Creating Joins and Unions	2-15
Creating If-Then-Else Expressions	2-15
Creating For-Let-Where-Order By-Return (FLWOR) Expressions.	2-17
Creating Typeswitch Expressions	2-19
Inserting XQuery Functions.	2-20
Inserting Expression Variables.	2-21
Viewing Schema Properties	2-22
Restricting Output of Optional Elements	2-23
Testing Data Transformations.	2-24
Graphical Features in Design View	2-27
Right-Click Menu Options.	2-27
Link Patterns	2-29
Link Colors	2-31
XML Global Elements, Global Types, Local Elements, and Attributes.	2-31

Examples: Data Transformation Using XQuery Mapper

Combining Data from Different Schemas.	3-2
Mapping Repeating Elements and Creating Joins.	3-5
Step 1. Create an XQuery File	3-6
Step 2. Add a Constraint	3-7
Step 3. Add Data to Return Element	3-9
Step 4. Add Function to Calculate Value of Quote	3-10
Step 5. Add a Constraint with Multiple Conditions.	3-12
Test the XQuery	3-13
Creating Unions.	3-14
Creating Repeating-Source to Nonrepeating-Target Transformations	3-17
Creating Nonrepeating-Source to Repeating-Target Transformation.	3-20
Creating Nested If-Then-Else Expressions	3-24

Step 1. Create the XQuery Transformation	3-24
Step 2. Create the First “If” Condition	3-25
Step 3. Create the First Nested If-Then-Else Condition.	3-27
Step 4. Create the Second Nested If-Then-Else Condition	3-28
Creating FLWOR Expressions	3-29
Using Recursive Schemas	3-33
Grouping Data by Key Fields	3-35

A. Upgrading XQuery Code

Upgrading Inline XQuery Code.	A-1
Upgrading XQuery Files	A-4

Introduction

Oracle XQuery Mapper is a graphical mapping tool that enables you to transform data between XML, non-XML, and Java data types, allowing you to integrate heterogeneous applications rapidly. For example, you can package data transformations in Oracle WebLogic Integration (WLI) as controls and reuse the controls in multiple business processes and applications. You can also use .xq files created in XQuery Mapper as resources in Oracle Service Bus.

The output of XQuery Mapper is a query in the XQuery language, which is defined by the World Wide Web Consortium (W3C). For more information about W3C and the XQuery language, see <http://www.w3.org/XML/Query/>.

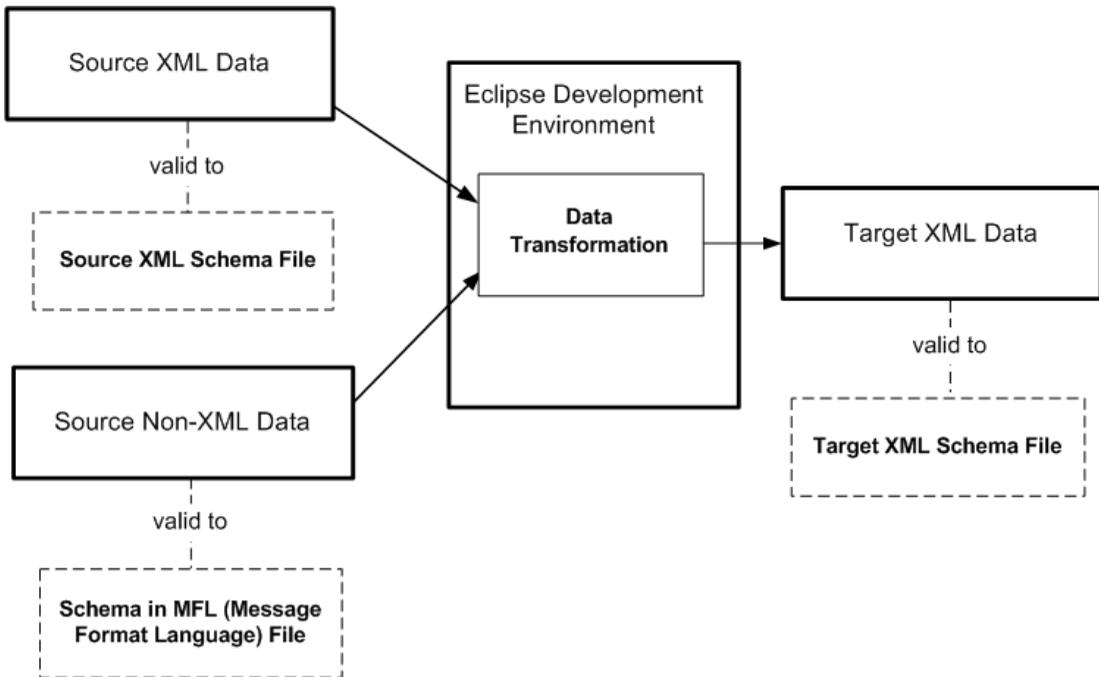
Overview of XQuery Mapper

You can use XQuery Mapper to transform data between XML, non-XML, and Java data types. For example, XML data that is valid against one schema can be converted to XML that is valid against a different schema. The data can be based on XML schemas, Web Service Definition Language (WSDL) file, and Message Format Language (MFL) files.

When you select the **Simple** source type, you can transform standard schema types, such as `boolean`, `byte`, `double`, `float`, `int`, `long`, `short`, `String`, and `Date`, to any other required target data format.

A data transformation can have multiple input types, but only one target type. For example, data can be transformed from two sources to one target, as shown in the following figure.

Figure 1-1 Data Transformation from Multiple Sources to One Target



Support for XQuery 2002 and 2004

WLI supports data transformation for the following versions of XQuery:

- XQuery 2004: Graphical design view (XQuery Mapper), source view, and test view.
- XQuery 2002: Source view and test view.

Note: For XQuery 2002-compliant XQuery files, the source view does not show compilation errors.

When you open an XQuery 2002-compliant XQuery file, it opens automatically in the **XQuery 2002 Transformation Editor**, which has **Source** and **Test** views, but no **Design** view.

Restrictions Applicable to the XQuery Test View

Table 1-1 Restrictions Applicable to the XQuery Test View

Restriction	Applicable to XQ2002?	Applicable to XQ2004?
If an XQuery calls a Java user function, the Java method must be static.	Yes	Yes
If an input Java type argument to an XQuery is an abstract class or an interface, the test view can not process it. An error message is displayed in the results view.	Yes	No
The input Java type and its member variables (except those of type primitive, String, <code>java.sql.Date</code> and <code>java.util.Date</code>) must follow the standard Java Bean guidelines.	Yes	No

Introduction

Transforming Data Using XQuery Mapper

You can use the graphical interface of XQuery Mapper to create data transformations, by mapping elements in source schemas to elements in a target schema. XQuery Mapper generates an XQuery, which is saved as an `.xq` file.

The procedure to transform data using XQuery Mapper is described in the following sections:

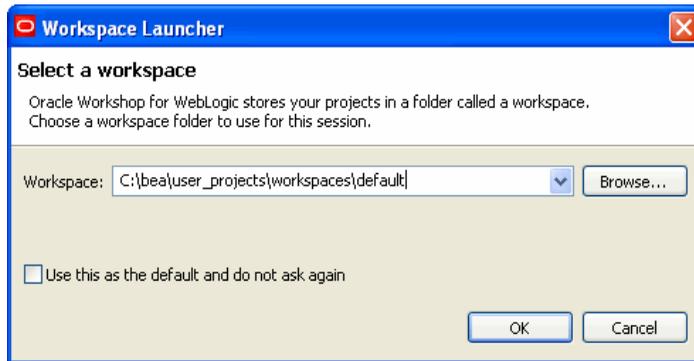
- [Launching XQuery Mapper](#)
- [Creating the XQuery Mapper Sample Project](#)
- [Creating an XQuery Mapper Project](#)
- [Importing and Creating Schema Files](#)
- [Selecting Source and Target Data Types](#)
- [Creating Data Transformations](#)
- [Editing Data Transformations](#)
- [Testing Data Transformations](#)
- [Graphical Features in Design View](#)
- [XML Global Elements, Global Types, Local Elements, and Attributes](#)

Launching XQuery Mapper

1. Choose **Start > Programs > Oracle WebLogic > Workshop for WebLogic 10gR3**.

The **Workspace Launcher** dialog box is displayed.

Figure 2-1 Workspace Launcher



2. In the **Workspace** field, specify the folder in which the project files must be stored.

Note: If required, select the **Use this as the default and do not ask again** check box, and click **OK**.

3. Open the XQuery transformation perspective by choosing **Window > Open Perspective > XQuery Transformation** from the Workshop menu.

The XQuery transformation perspective launches automatically when you open an XQuery file. If, however, XQuery Mapper is open and no XQuery file is open, you must launch the XQuery transformation perspective manually.

Creating the XQuery Mapper Sample Project

The XQuery Mapper sample project includes sample schema and XML files, which you can use to create XQuery transformations as described in [Chapter 3, “Examples: Data Transformation Using XQuery Mapper.”](#)

You can create the sample project by using a wizard from within the XQuery transformation perspective.

1. From the Workshop menu bar, choose **File > New > Other > WebLogic Integration > Tutorial: XQuery Transformation**, and click **Next**.

2. Enter a name for your sample project and click **Finish**.

The sample project is created and displayed in the **Navigator** view. The project contains the following folders:

- **schemas** folder: Contains the XML schema (.xsd) files of the sample project.
- **XML** folder: Contains test XML files required by some of the samples.
- **XQueryTransformations** folder: This is the folder in which you will create the XQuery files for the samples.

To learn more about creating projects and importing the files you need for those projects, see the following topics:

- [Creating an XQuery Mapper Project](#)
- [Importing and Creating Schema Files](#)
- [Chapter 3, “Examples: Data Transformation Using XQuery Mapper”](#)

Note: You can import project-specific XML schemas, Web Service Definition Language (WSDL) files, and Message Format Language (MFL) files from any location. Before you import the files, it is recommended that you create a folder structure that meets your business needs. For more information, see [“Importing and Creating Schema Files” on page 2-4](#).

Creating an XQuery Mapper Project

1. Switch to the XQuery transformation perspective by choosing **Window > Open Perspective > XQuery Transformation**.
2. Choose **File > New > Project**.
The **New Project** wizard is displayed.
3. Choose **General > Project** and click **Next**.
4. Enter a name for the project.
5. Ensure that the **Use default location** check box is selected.
6. Click **Finish**.

Importing and Creating Schema Files

Schema files can be created in and imported from any location. The following schema types are supported:

- **XSD (XML Schema Definition)**: XML schemas describe and constrain data in XML files. Multiple namespaces are supported in XQuery Mapper. For example, you can transform data from two source XML files that are valid against a specific namespace to an XML file that is valid against another namespace.
- **WSDL (Web Service Definition Language)**: XML schema defined in the WSDL file can be used in data transformations.
- **MFL (Message Format Language)**: MFL files describe and constrain data in non-XML files (COBOL copybooks and C structure definitions, for example). The namespace of the MFL elements is derived from the name of the MFL file.

MFL files are created using the Format Builder tool and have the `.mfl` extension. For more information, see the [Format Builder Online Help](#).

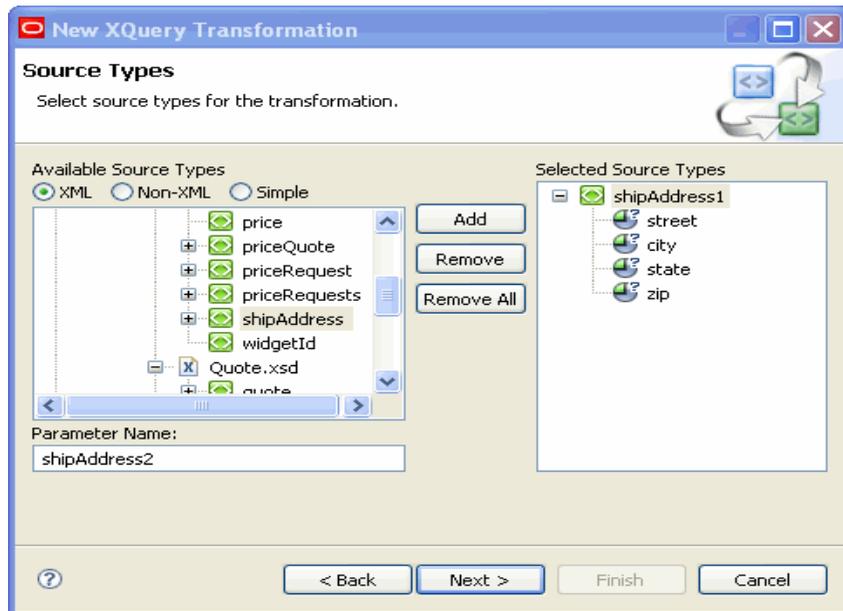
Importing XML Schemas and MFL Files

1. Switch to the XQuery transformation perspective by choosing **Window > Open Perspective > XQuery Transformation**.
2. In the **Navigator** view, select the project into which you want to import the XML schema or MFL file.
3. Choose **File > Import**.
The **Import** wizard is displayed.
4. You can import XML schemas and MFL files from a variety of sources. Select the appropriate source and click **Next**.
5. Browse for and select the required file, and click **Finish**.

After the schemas files or MFL files are imported, they are available in the **New XQuery Transformation** wizard.

The following figure shows how the imported XML schemas are displayed in the **New XQuery Transformation** wizard.

Figure 2-2 New XQuery Transformation

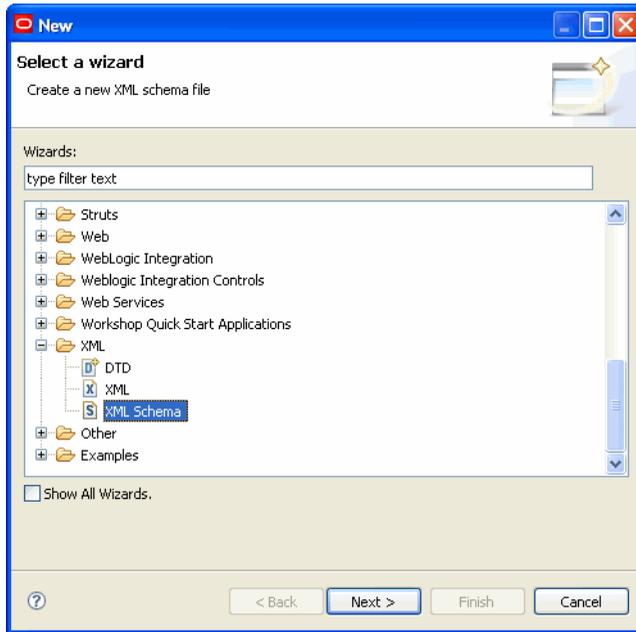


Creating XML Schemas

You can create XML schemas by using the XML Schema Editor.

1. Switch to the XQuery transformation perspective by choosing **Window > Open Perspective > XQuery Transformation**.
2. In the **Navigator** view, select the project in which you want to create the schema files.
3. Select **File > New > Other ...**
The **New** screen is displayed.

Figure 2-3 Select a Wizard



4. Expand the **XML** node.
5. Select **XML Schema** and click **Next**.
6. Select the parent folder in which you want to create the schema file.
7. Enter a name for the schema file and click **Finish**.

The schema file is created in the specified project. You can now specify the details of the schema and save the file.

For information about using the XML Schema Editor, see *Introduction to the XSD Editor* at <http://www.eclipse.org/webtools/community/tutorials/XMLSchemaEditor/XMLSchemaEditorTutorial.html>.

Creating XML Files from XML Schemas

1. Switch to the XQuery transformation perspective by choosing **Window > Open Perspective > XQuery Transformation**.
2. In the **Navigator** view, select the project in which you want to create the schema files.

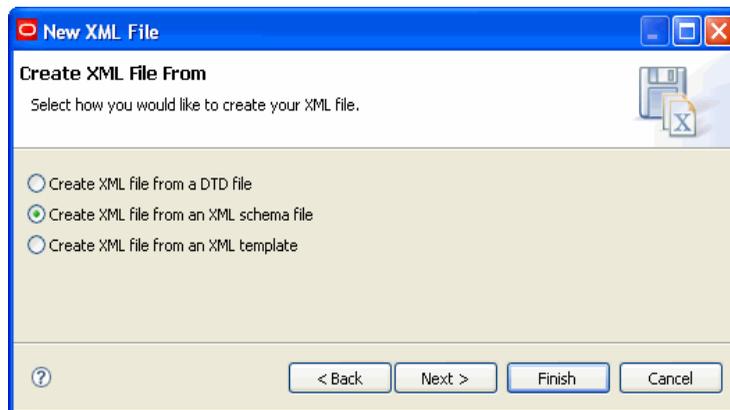
3. Select **File > New > Other ...**

The **New** screen is displayed.

4. Expand the **XML** node.
5. Select **XML** and click **Next**.

The **Create XML File** dialog box is displayed.

Figure 2-4 Create XML File



6. Select the **Create XML file from an XML schema file** option and click **Next**.
7. Select the parent folder in which you want to create the XML file.
8. Enter a name for the XML file and click **Next**.
9. Select the XML schema based on which you want to create the XML file and click **Next**.
10. Select the root element of the XML file and click **Finish**.

The XML file is created in the specified folder. You can now specify the details of the file and save it.

For information about using the XML Editor, see *Creating XML files Tutorial* at <http://www.eclipse.org/webtools/community/tutorials/XMLWizards/XMLWizards.html>.

Creating WSDL Files

1. Switch to the XQuery transformation perspective by choosing **Window > Open Perspective > XQuery Transformation**.

2. In the **Navigator** view, select the project in which you want to create the WSDL files.
3. Select **File > New > Other ...**
The **New** wizard is displayed.
4. Expand the **XML** node, select **WSDL**, and click **Next**.
5. Select the parent folder in which you want to create the WSDL file.
6. Enter a name for the WSDL file and click **Next**.
7. Enter the target namespace and prefix of the WSDL file.
8. If required, select the **Create WSDL Skeleton** check box.
9. Select the protocol and the binding option and click **Finish**.

The WSDL file is created in the specified project.

For information about using the WSDL Editor, see *Introduction to the WSDL Editor* at:
<http://www.eclipse.org/webtools/community/tutorials/WSDLEditor/WSDLEditorTutorial.html>.

Creating MFL Files

You can create MFL files by using the Format Builder tool (**Start > Programs > Oracle WebLogic Integration > WebLogic Integration 10gR3 > Format Builder**).

For more information about the Format Builder tool, see *Format Builder Online Help*.

Selecting Source and Target Data Types

Before you create a data transformation, you must define the source target types and a target data type. The source and target types can be non-XML, XML, and simple data types.

1. Select the project for which you want to select source and target data types.
2. Right-click and choose **New > XQuery Transformation**.
The **New XQuery Transformation** wizard is displayed.
3. Enter a name for the **.xq** file.
4. Click **Next**.

The **Source Types** dialog box is displayed.

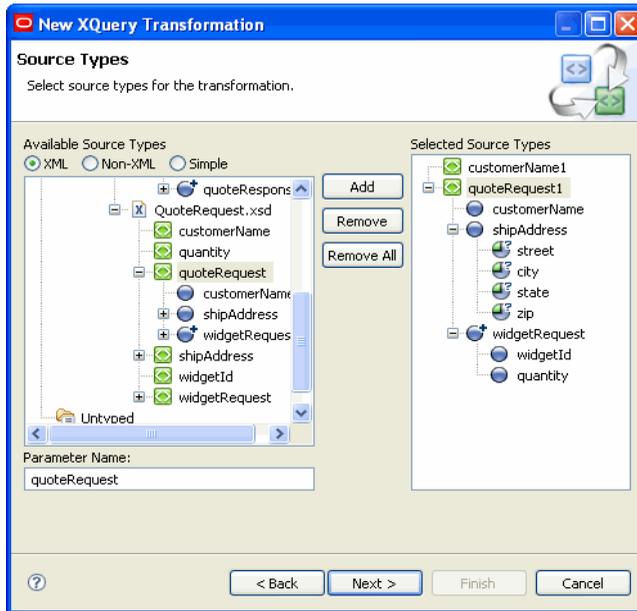
5. In the **Available Source Types** pane select the source data types.
 - If the source is XML data, choose **XML**.
 - If the source is MFL data, choose **Non-XML**.
 - If the source data is of standard types, such as `boolean`, `String`, and `int`, then choose **Simple**.
6. Select the required source data elements.

Note: For schemas to be displayed in the **Available Source Types** and **Available Target Types** pane, the XML and non-XML files that contain these schemas must first be imported into or created in Workshop.

The **Available Source Types** and **Available Target Types** panes show only schemas that exist in the `schemas` folder of the XQuery Transformation project. If you prefer to keep your schemas in any other folder, you must specify the path to that folder in the XMLBeans settings for the project (by choosing **Project** > **Properties**, the **XMLBeans** page, and then the **Source Paths** tab).

For example, to add input data from `schemas/Dates.xsd`, select the `date` element in the schema as the input element and click **Add**, as shown in the following figure.

Figure 2-5 Selecting Source Types



The elements and attributes that make up the selected element are displayed in the **Selected Source Types** pane.

7. After selecting the required source types, click **Next**.

The **Target Types** dialog box is displayed.

8. In the **Available Target Types** pane, select the target data type, and click **Add**.

The elements and attributes that make up the selected element are displayed in the **Selected Target Type** pane.

Note: You can specify only one target data type.

9. Click **Finish**.

The **.xq** file is displayed in the **Design** view. It shows the source and target data types that you selected.

Creating Data Transformations

You can perform the following types of data transformations:

- Basic element transformations: Mapping a source element to a target element.
- Basic attribute transformations: Mapping a source attribute to a target attribute.
- Complex transformations: Mapping a complex source (for example, a repeating element) to a complex target (for example, a nonrepeating element).

Creating Basic Element Transformations

Basic element transformation involves mapping a source element to a target element. The source and target elements may have the same name, type, or scope.

The following are some examples of the types of basic element transformation that you can perform:

- **Element to element:** A source element is mapped to a target element.
- **Element combination:** Multiple source elements are combined to create a single target element.
- **Element explosion:** XQuery string functions are exploded from a single source element to multiple target elements.

Prerequisite

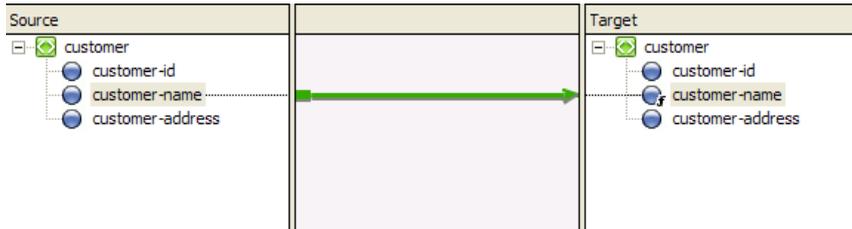
The XQuery file is created as described in [“Selecting Source and Target Data Types” on page 2-8](#).

Creating Element-to-Element Links

1. Select the project for which you want to create element-to-element links, and open the XQuery file in which the transformation must be stored.
2. Drag the required element from the **Source** pane to the target element in the **Target** pane.

For example, to create a link between the `customer-name` element in the source schema and the `customer-name` element in the target schema, drag `customer-name` from the **Source** pane to the **Target** pane. An arrow connects the two elements, as shown in the following figure.

Figure 2-6 Element-to-Element Links



Note: While dragging from the **Source** pane to the **Target** pane, a dashed line appears temporarily between the two elements. For more information about link patterns, [“Link Patterns” on page 2-29](#).

3. After creating the required element-to-element links, save the changes.

Creating Basic Attribute Transformations

Basic attribute transformation involves mapping a source attribute to a target attribute. The source and target attributes may have the same name, type, or scope.

The following are some examples of basic attribute transformations:

- **Element to attribute:** A source element is mapped to a target attribute.
- **Attribute to element:** A source attribute is mapped to a target element.
- **Attribute to attribute:** A source attribute is mapped to a target attribute of the same name.

Prerequisite

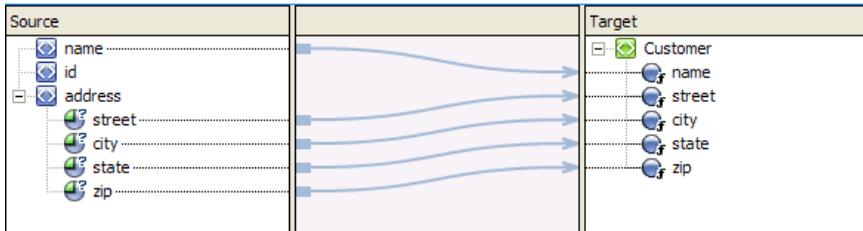
The XQuery file is created as described in [“Selecting Source and Target Data Types” on page 2-8](#).

Creating an Attribute-to-Element Link

1. Select the project for which you want to create attribute-to-element links, and open the XQuery file in which the transformation must be stored.
2. Drag the required attribute from the **Source** pane to the appropriate element in the **Target** pane.

For example, to create a link between the `street` attribute of the `address` element in the source schema and the `street` element of the target schema, drag the `street` attribute from the **Source** pane to the **Target** pane, as shown in the following figure.

Figure 2-7 Attribute-to-Element Link



3. After creating the required links, save the changes.

Similarly, you can create element-to-attribute and attribute-to-attribute links.

Creating Complex Transformations

Complex transformations involve mapping a complex source (for example, a repeating element) to a complex target (for example, a nonrepeating element). The following are some examples of complex transformations:

- **Repeating group to repeating group:** The source contains a variable number of instances of a group of elements; each source instance is mapped to an instance of the target group.
- **Repeating group to nonrepeating element:** The source contains a variable number of instances of a group of elements; each source group is mapped to an instance of the target element.

Prerequisite

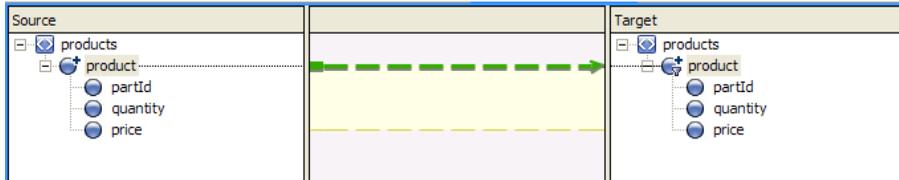
The XQuery file is created as described in “[Selecting Source and Target Data Types](#)” on page 2-8.

Creating a Complex Transformation

1. Select the project for which you want to create the links, and open the XQuery file in which the transformation must be stored.
2. Drag the required element or attribute from the **Source** pane to the appropriate element or attribute in the **Target** pane.

For example, to create a link between `product` (a repeating group in the source schema) and `product` (a repeating group in the target schema), drag `product` from the **Source** pane to the **Target** pane, as shown in the following figure.

Figure 2-8 Repeating-Group-to-Repeating-Group Link



3. After creating the required links, save the changes.

Editing Data Transformations

After creating a data transformation in the **Design** view, you can add, change, and delete XQuery code either by editing code directly in the **Source** view or by adding complex expressions in the **Design** view.

Note: For information about the XQuery language, see <http://www.w3.org/XML/Query>.

This section contains information about the following topics:

- [Viewing and Editing XQuery Files](#)
- [Creating Joins and Unions](#)
- [Creating If-Then-Else Expressions](#)
- [Creating For-Let-Where-Order By-Return \(FLWOR\) Expressions](#)
- [Creating Typeswitch Expressions](#)
- [Inserting XQuery Functions](#)
- [Inserting Expression Variables](#)
- [Viewing Schema Properties](#)

Viewing and Editing XQuery Files

1. Select the project containing the XQuery file that you want to edit.
2. Double-click the XQuery file.

Note: If the XQuery file is XQuery 2002-compliant, it opens automatically in the XQuery 2002 Transformation Editor, which has only **Source** and **Test** views. For more information, see “[Support for XQuery 2002 and 2004](#)” on page 1-2.

3. Select the **Source** view.

The XQuery code is displayed. Invalid code is underlined in red.

4. Make the required changes.

Note: If necessary, you can delete the data transformations in the **Source** view by deleting all the code within the function except the root element.

5. Save the changes.

Creating Joins and Unions

The **Constraints** view in the XQuery transformation perspective allows you to constrain or manipulate the relationship between source and target repeating elements.

The following **Constraint Type** options are available in the **Constraints** view:

- **Repeatability/Join** option

When you create transformations between repeating elements, **for** loops are generated to iterate through the repeating elements. You can limit or constrain the target repeating elements by adding **where** clauses to the **for** loops in the **Where Clause** pane of the **Constraints** view.

You can create complex conditions (joined by OR or AND operators) for the **where** clause, as shown in the following code example:

```
((data($PurchaseOrderDoc/partId) > 200 and
data($PurchaseOrderDoc/partId) <= 400))
```

At run time, the **for** loop iterates over only those repeating elements that fulfill the complex condition.

For a detailed example on using the **Constraints** view, see [“Creating Repeating-Source to Nonrepeating-Target Transformations”](#) on page 3-17.

- **Union**

See [“Creating Unions”](#) on page 3-14.

Creating If-Then-Else Expressions

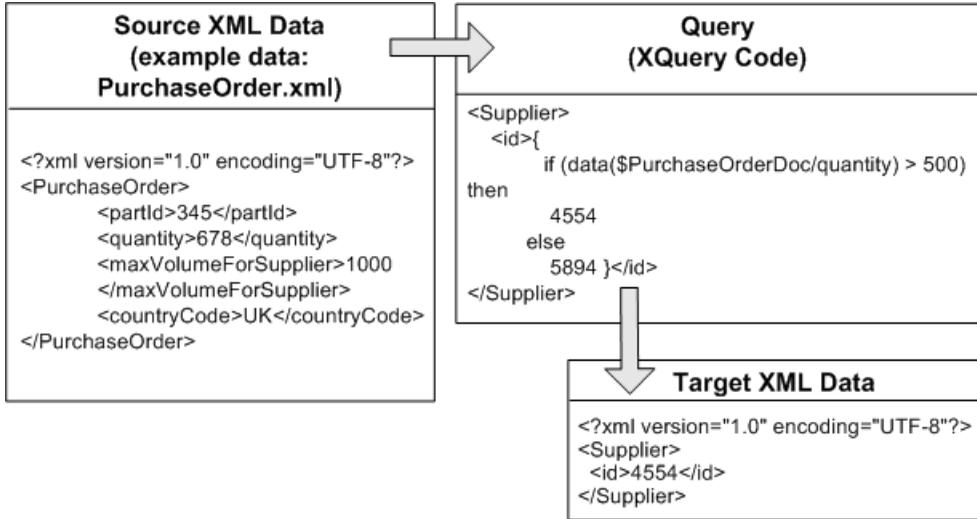
The **Target Expression** view allows you to create if-then-else expressions.

When a query that contains an if-then-else expression is executed, the conditions that make up the **if** expression are evaluated. Depending on the result, different values are returned for the target node.

Figure 2-9 shows XQuery code that can be used to implement the following logic:

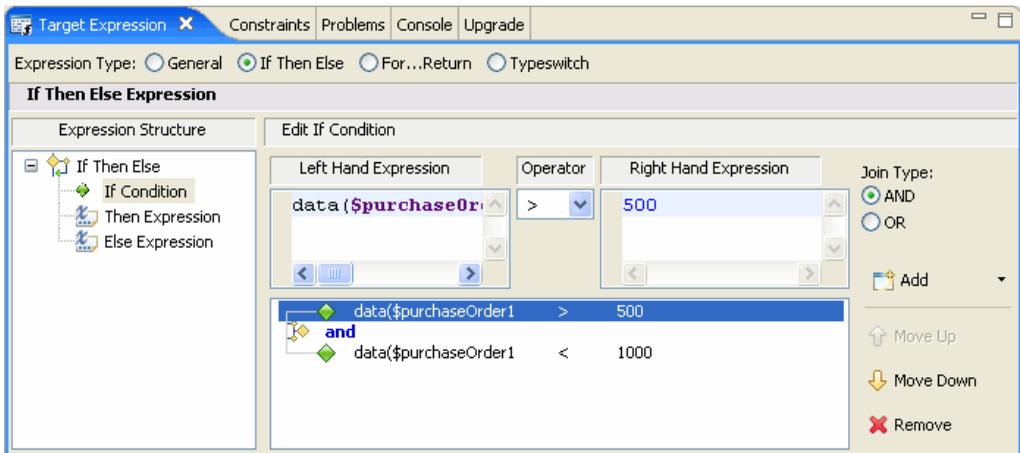
- If the value of the **quantity** source node is more than 500, then return 4554 as the value of the **ID** target node
- If **quantity** is less than or equal to 500, then return 5894 as the **ID**.

Figure 2-9 Example of If-Then-Else Expression



You can add multiple expressions to the **If** condition, as shown in the following figure.

Figure 2-10 If-Then-Else Expression in Target Expression View



You can change the position of a condition by selecting it and then clicking the **Move Up** or **Move Down** button. You can also remove a condition by selecting it and then clicking **Remove**.

Note: In the **Edit If Condition** pane, even if you remove all the expressions by using the **Remove** button, the if-then-else expression is not removed entirely in the **Source** view. The expressions associated with the **if** condition are removed, but the **then** and **else** expressions are retained, as shown in the following listing.

```
<ns0:partId>
  {
    if (fn:boolean("true"))
      then 4554
      else 5894
  }
</ns0:partId>
```

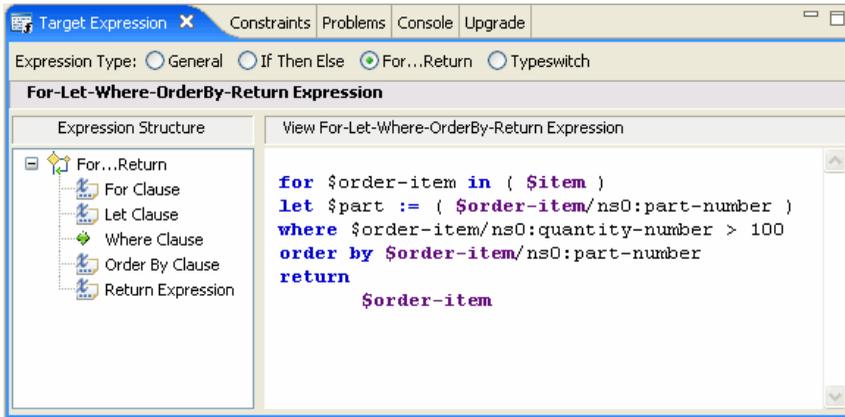
The XQuery always returns the **then** expression. The **else** expression is retained in the code so that you can reuse it in the future, if required.

For more information, see [“Creating Nested If-Then-Else Expressions”](#) on page 3-24.

Creating For-Let-Where-Order By-Return (FLWOR) Expressions

The **Target Expression** view allows you to create FLWOR expressions, as shown in the following figure.

Figure 2-11 FLWOR Expression



The following table describes the components of FLWOR expressions.

Table 2-1 Clauses of FLWOR Expressions

Component	Description	Optional or Mandatory	Allowed Nested Expressions
For clause	The For clause iterates over a sequence of input items and returns a value for each item.	At least one For or Let clause	<ul style="list-style-type: none"> If-Then-Else FLWOR Typeswitch
Let clause	The Let clause declares a variable and assigns a value to the variable.	At least one For or Let clause	<ul style="list-style-type: none"> If-Then-Else FLWOR Typeswitch
Where clause	The Where clause specifies the basis for filtering input data. It is similar to the If clause.	Optional	
Order By clause	The Order By clause specifies the basis for sorting the output of the query.	Optional	<ul style="list-style-type: none"> If-Then-Else FLWOR Typeswitch
Return expression	The Return expression defines the output of the query.	Mandatory (only one)	<ul style="list-style-type: none"> If-Then-Else FLWOR Typeswitch

You can insert nested expressions under For, Let, Order By, and Return by right-clicking on them and selecting the required expression from the menu.

Note: In the **Design** view, if you create a link between repeating elements, a For...Return expression (implicit FLWOR expression) is generated automatically in the **Source** view. You can add Where clauses to this FLWOR expression by using the **Constraints** view, but you cannot add Let and Order By clauses. Implicit FLWOR expressions are not shown in the **Target Expression** view.

For more information, see [“Creating FLWOR Expressions” on page 3-29](#).

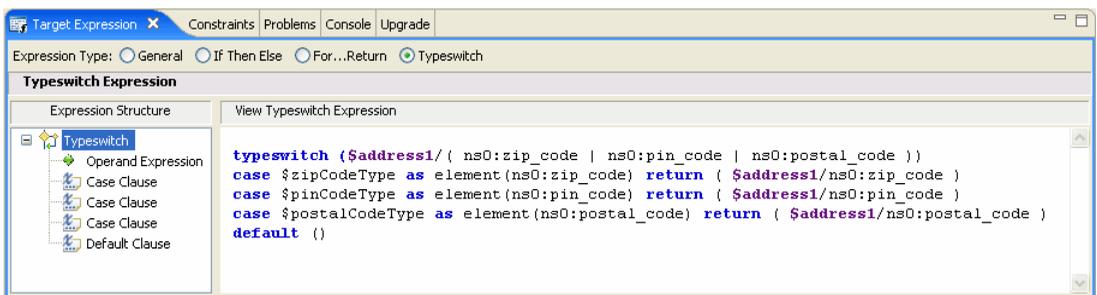
Creating Typeswitch Expressions

Typeswitch expressions may be required in the following situations:

- When an XML schema contains a <choice> element, which allows only one of the elements defined in the <choice> declaration to be present in the containing element, you can use a typeswitch expression to determine the type of the <choice> element that is present in the source XML file and, accordingly, return a value.
- When an XML schema contains a substitution group, which allows one element to be substituted for another, you can use a typeswitch expression to determine the type of the element that is actually in the source XML file and, accordingly, return a value.

The **Target Expression** view lets you create typeswitch expressions, as shown in the following figure.

Figure 2-12 Typeswitch Expression



The following table describes the components of typeswitch expressions.

Table 2-2 Clauses of Typeswitch Expressions

Component	Description	Optional or Mandatory	Allowed Nested Expressions
Operand expression	The Operand is the expression for which the type is to be evaluated.	Mandatory (only one)	
Case clause	Each Case clause specifies the name of the type to be evaluated and the expression to be returned if the evaluation is true.	Mandatory (at least one)	<ul style="list-style-type: none"> • If-Then-Else • FLWOR • Typeswitch
Default clause	The Default clause specifies the return expression that must be used if the value of the operand expression matches none of the types specified in the Case clauses.	Mandatory (only one)	<ul style="list-style-type: none"> • If-Then-Else • FLWOR • Typeswitch

Inserting XQuery Functions

A set of standard W3C XQuery functions and operators is provided in XQuery Mapper. You can add standard XQuery or user-defined functions in XQuery files and data transformation files. For example, you can use the `upper-case` XQuery `String` function to convert the characters in an XML String value to uppercase characters.

Note: For more information about XQuery 1.0 and XPath 2.0 functions and operators (W3C Working Draft 23 July 2004), see <http://www.w3.org/TR/2004/WD-xpath-functions-20040723/>.

1. Switch to the XQuery transformation perspective by choosing **Window > Open Perspective > XQuery Transformation**.

The **Expression Functions** view is part of the XQuery transformation perspective.

2. Open the XQuery file in which you want to insert XQuery functions.
3. In the **Design** view, select or create a link to add the function call.

The link becomes green.

4. Select the **Target Expression** view.

If the **Target Expression** view is not visible, choose **Window > Show View > Target Expression**.

In the **General Expression** pane, the XQuery code linking the selected target and source node is displayed and is selected. Keep this selected for the next step.

5. Delete the existing XQuery code in the **General Expression** pane.
6. Find the function that you want to insert in the **Expression Functions** view.
For this example, from the **String Functions** folder, select the **upper-case** function, which converts all the characters of the source element to upper case.
7. Drag the **upper-case** function to the **General Expression** pane.
Leave the parameter of the selected function (the **\$string-var** parameter of the **upper-case** function in this example) in the **General Expression** pane selected.
Note: XQuery functions that are defined by Oracle (example: **trim-left**) are prefixed with **fn-bea:**.
Note: XQuery functions that are not listed in the **Expression Functions** view but defined in the XQuery specification, can be used with the **fn:** prefix.
8. Select a source parameter for the function using one of the following options:
 - From the **Source** pane of the **Design** view, select a source element, drag it to the **General Expression** pane and drop it over the **\$string-var** parameter.
 - From the **Expression Variables** view, select a source variable, drag it to the **General Expression** pane and drop it over the **\$string-var** parameter.
9. After inserting the required functions and assigning parameters to the functions, click **Apply** in the **General Expression** pane.

Inserting Expression Variables

The variables (and their subelements) that you can use in an XQuery are displayed in the **Expression Variables** view.

Note: If the **Expression Variables** view is not displayed, choose **Window > Show View > Expression Variables**.

The following types of variables are displayed in the **Expression Variables** view:

- **Source:** The variables displayed under the **Source** node in the **Expression Variables** view are those that are selected for the transformation in the **Source Types** dialog box of the **New XQuery Transformation** wizard.

- **Structural Link:** The variables displayed under the **Structural Link** node in the **Expression Variables** view are the loop iteration variables that are associated with the XQuery `for` loops generated by structural links.

These variables are in scope for all the subelements of the node that has the structural link.

You can insert expression variables in the following ways:

- **Drag-and-drop**

Drag the variables or their subelements from the **Expression Variables** view, and drop them in the **Constraints** or **Target Expression** view.

- **Enter \$ and choose a variable from the pop-up menu**

- a. Enter the dollar symbol (\$) in the required text field. For example, enter \$ in the **General Expression** pane of the **Target Expression** view.

A pop-up menu containing a list of the available variables is displayed.

- b. Choose the required variable and then enter a forward slash (/).

If subelements exist, a pop-menu containing a list of the available subelements is displayed.

- c. Choose the required subelement and then enter a forward slash (/).

If further subelements exist, a pop-menu containing a list of the available subelements is displayed.

- d. Repeat step c until you finish entering the required variable.

Viewing Schema Properties

While editing an XQuery file in the **Design** view, you can view the schema properties of nodes in the current transformation in the **Properties** view, without opening the source and target XSD or MFL files.

To display the **Properties** view, choose **Window > Show View > Properties**.

- If you select an element or attribute in the **Source**, **Target**, or **Expression Variable** view, the associated schema properties are displayed in the **Properties** view.
- If you select a link, the schema properties of the target and source elements of the link are displayed in the **Properties** view.

- To deselect a link (including the target and source nodes of the link), click anywhere in the empty area of the pane between the **Source** and **Target** panes of the **Design** view.

Note: To change the schema properties of an element or attribute, edit the corresponding schema file (XSD for XML schema and MFL for non-XML schema).

Restricting Output of Optional Elements

If the target schema in a data transformation contains an optional element (`minOccurs="0"`), you can design the link to the element such that the element is included in the output XML file only if it contains a value (that is, the element is not empty in the source XML file).

Consider the source XML data in the following listing.

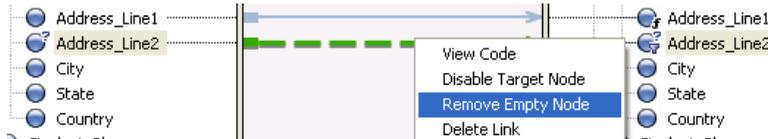
Listing 2-1 XML Data with Optional Element

```
<address>
<Address_Line_1>1 Elm Street</Address_Line_1>
<Address_Line_2/>
<City>San Jose</city>
<State>California</State>
<Country>US</Country>
</address>
```

The `Address_Line_2` element is optional and empty. If `Address_Line_2` is mapped to a corresponding element in the target schema, then, by default, the output XML file contains an empty `Address_Line_2` element.

You can restrict output of such optional elements by right-clicking on the link and selecting **Remove Empty Node**, as shown in the following figure.

Figure 2-13 Remove Empty Node



The XQuery code underlying the link is enclosed in an if-then-else expression that causes the target element to be produced only if the transformation results in a nonempty value.

You can remove the if-then-else expression by right-clicking on the link and selecting **Retain Empty Node**, as shown in the following figure.

Figure 2-14 Retain Empty Node



Note: The **Remove Empty Node** (or **Retain Empty Node**) option is displayed only when you right-click on a link to an optional target element.

Testing Data Transformations

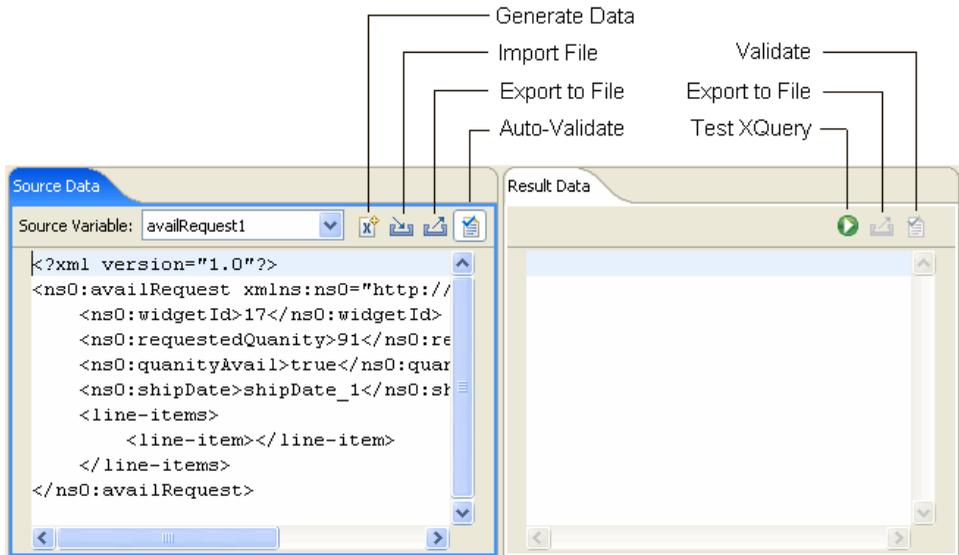
After creating an XQuery transformation in the **Design** view, you can test whether the expected XML or non-XML output is generated properly in the **Test** view.

You can use the autogenerated XML files or your own custom XML and non-XML files for testing the transformations.

Features of the Test View

The following figure shows the features of the **Test** view.

Figure 2-15 Test View



- **Source Variable**

The values available in the **Source Variables** drop-down list are based on the source XML schemas of the transformation that you are testing. When you select one of these schemas, an XML file is generated automatically and displayed.

Note: These XML files are not saved automatically; you can save them by clicking the **Export** icon.

To use a custom XML file (instead of the autogenerated XML files) or a non-XML file (such as MFL) for testing the transformation, you can import the file by clicking the **Import** icon.

- **Generate Data**

When you select the **Test** view, XQuery Mapper generates an initial set of sample data and displays it in the **Source Data** pane.

If you want to regenerate the sample data, click **Generate Data**. You might, for example, want to start testing afresh with new sample data if edits have resulted in XML data that is no longer valid for the input schema.

If the XQuery that you want to test has multiple source types, you can generate sample data for each source type by selecting the required source type in the **Source Variable** drop-down list and then clicking the **Generate Data** icon.

You can also manually edit the generated XML data.

Note: For complex input schemas, the generated XML data may not comply with the schema. Validation errors are underlined in yellow in the **Source Data** tab. When you place the mouse pointer over an error, details of the error are displayed. You can correct the generated XML data to make it comply with the input schema.

- **Import File**

You can import data from an XML (or non-XML) file and test the transformation using that data.

In addition, if the XQuery that you want to test has multiple source types, you can import sample data for each source type by selecting the required source type in the **Source Variable** drop-down list and then clicking the **Import File** icon.

If the **Auto Validate** option is selected, then, when you import data, the data is validated against the associated schema. Errors are underlined in yellow. When you place the mouse pointer over an error, details of the error are displayed.

Note: You can import XML data for global types and local elements, but global types and local elements are not validated and no errors or warnings are reported for invalid data. For more information, see [“XML Global Elements, Global Types, Local Elements, and Attributes” on page 2-31](#).

- **Export to File:** You can save the data in the **Source Data** pane or the results of the transformation in the **Result Data** pane to an XML file.
- **Test XQuery:** When you select this option, the XQuery is executed on the data in the **Source Data** pane and the result of the transformation is displayed in the **Results Data** pane.
- **Options for validating at design time**

The **Auto Validate** option in the **Source Data** pane and the **Validate** option in the **Result Data** pane are enabled only if the source parameter or result data is an XML global element.

The validation options are not available for the following data types:

- Typed non-XML

Note: In any case, untyped non-XML (raw) data cannot be used in data transformations.

- XML global type
- XML local element

For more information, see [“XML Global Elements, Global Types, Local Elements, and Attributes”](#) on page 2-31.

Selecting the **Auto Validate** option causes the source data to be validated automatically against the source schema every time the data is changed.

You can use the **Validate** option to validate the result of the transformation against the target schema.

Note: The validation at design time in the **Test** view is not the same as the schema validation that occurs at run time. Validation at design time does **not** modify the resulting XML document, but it does check for existence of elements and attributes that are defined as required in the schema.

Related Topics

[Restrictions Applicable to the XQuery Test View](#)

Graphical Features in Design View

This section provides information to help you use the graphical features of XQuery Mapper and interpret the graphical representations in the **Design** view of XQuery Mapper.

Right-Click Menu Options

The following table lists the options available in the **Design** view of XQuery Mapper when you right-click on a link or element.

Table 2-3 Right-Click Menu Options

Menu Option	Appears When You ...	Result ...
View Code	Right-click on any link or target element	The view switches to the Source view, and the XQuery code for the link is selected.
Create Constant	Right-click on any simple-type target element	Lets you assign a constant value to the target element.

Table 2-3 Right-Click Menu Options

Menu Option	Appears When You ...	Result ...
Disable Target Node	Right-click on any link or target element	Blocking XQuery code is added around the XQuery code of the selected link. The blocking code prevents the XQuery code for the link from being executed at run time. Note: The Disable Target Node menu option is not available for the root node of the target type.
Enable Target Node	Right-click on a disabled link or target element	The blocking XQuery code is removed from around the selected link so that, at run time, the XQuery code for the link is executed.
Remove Empty Node	Right-click on a link to an optional target element.	Surrounds the link with an if-then-else expression that causes the target element to be produced only if the transformation results in a nonempty value.
Retain Empty Node	Right-click on a link (to an optional target element) for which you selected the Remove Empty Node option earlier.	Removes the if-then-else expression that causes the target element to be produced only if the transformation results in a nonempty value.
Redefine Wildcard Node	Right-click on a wildcard (any type) source or target element	Lets you define a specific data type for a wildcard element. Note: After you redefine a wildcard node (that is, define a specific data type), you cannot use Ctrl+Z to revert to the any type.
Revert to Wildcard Node	Right-click on a wildcard (any type) source or target element, for which you defined a specific data type earlier by using the Redefine Wildcard Node option.	Changes the data type of the element from the specific type that you defined earlier to wildcard (any type).
Induce Map	Right-click on a structural link.	Data links or data structural links are created between the child nodes of the selected structural link if source and target child elements of the link are the same subschema type. Note: For the Induce Map option to create child links, the target and source child elements must have the same name and data type, and must be in the same order.

Table 2-3 Right-Click Menu Options

Menu Option	Appears When You ...	Result ...
Delete Link	Right-click on any link.	<ul style="list-style-type: none"> • Design view: The link is deleted. • Source view: The XQuery code underlying the link is deleted.
Delete All Links	Right-click anywhere in the empty pane between the Source and Target panes.	<ul style="list-style-type: none"> • Design view: The lines representing the transformation between source and target elements/attributes are deleted. • Source view: The generated XQuery code is deleted. <p>Note: Right-clicking anywhere in the empty pane between the Source and Target panes causes all the nodes to be deselected.</p>

Link Patterns

Links in XQuery Mapper are shown in different colors and patterns to help you distinguish easily between different link types. The following table describes the graphical representation of the links that you create in XQuery Mapper.

Table 2-4 Link Patterns

Link Type	Description	Link Currently Selected?	Representation
Data Link	A link that converts the value of the source node directly to the value of the target node.	Not Selected	
		Selected	

Table 2-4 Link Patterns

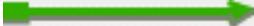
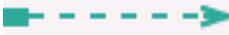
Link Type	Description	Link Currently Selected?	Representation
Implied Link	A data link for which you modified the underlying XQuery code.	Not Selected	
	A link for which the XQuery code cannot be interpreted by the XQuery Mapper. Examples: <ul style="list-style-type: none"> A link for which you inserted the fn:upper-case XQuery function by using the General Expression section of the Target Expression view. The data links generated between a second set of child nodes when a union constraint is applied to a set of two structural links. The child nodes must be of the same subschema. 	Selected	
Structural Link	A link between two parent structures and that does not map data directly.	Not Selected	
		Selected	
Data Structural Link	A data structural link is a combination of a data link and a structural link. Example: A link between the optional child nodes of a repeating element.	Not Selected	
		Selected	
Constraint Link	A link that constrains or limits the resulting data of a join between source parent structures. The constraint link is created with two source nodes. Example: A a join between two source repeating elements to return the data only when the values of a particular source element are equal to each other.	Not Selected	
		Selected	

Table 2-4 Link Patterns

Link Type	Description	Link Currently Selected?	Representation
Copy Link	<p>A link between two identical schema substructures. At run time, the source data is copied directly, as a block, to the target data.</p> <p>A copy link is also generated when you map an untyped XML node to a typed XML complex-type node.</p>	Not Selected	
		Selected	

Link Colors

When you drag a node from the **Source** pane to the **Target** pane, a temporary link (dashed line) appears between the two nodes. The color of the line changes depending on the compatibility between the source and target nodes.

- **Red:** The link cannot be created between the source node and the target node because the data type of the target node cannot be converted to the data type of the source node. For example, a node of XML String data type cannot be converted to an XML repeating node. An error message is displayed when you drag the source node over the target node.
- **Orange:** The link can be created between the source and target nodes, but the data types are not completely compatible. A warning message describing the incompatibility or any necessary conversion is displayed when you drag the source node over the target node.
- **Green:** The link can be created between the source and target nodes. The data type of the target node is compatible with the data type of the source node.

When you finish creating the link, a dotted or dashed line (depending on the source and target nodes) is displayed.

XML Global Elements, Global Types, Local Elements, and Attributes

An XML schema type or element is considered global if it is a direct child of the `schema` element and local if it is not a direct child of the `schema` element (that is, the element is nested within another element).

The following example XML schema illustrates this difference.

Listing 2-2 XML Schema with Global and Local Elements

```
<?xml version="1.0"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
targetNamespace="http://www.acme.org/globalExample"
xmlns="http://www.acme.org/globalExample"
elementFormDefault="qualified"
attributeFormDefault="unqualified">
  <xs:element name="globalElement">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="localElement"
          minOccurs="1" maxOccurs="1"
          type="xs:string" />
      </xs:sequence>
      <xs:attribute name="attribute"
        type="xs:string" use="required"/>
    </xs:complexType>
  </xs:element>
  <xs:complexType name="globalType">
    <xs:sequence>
      <xs:element name="anotherLocalElement"
        minOccurs="0" maxOccurs="unbounded"
        type="xs:string" />
    </xs:sequence>
  </xs:complexType>
</xs:schema>
```

In the preceding XML schema, **globalElement** is global because it is a direct child of the **schema** element, whereas **localElement** is local because it is a child of **globalElement**.

You can also define a global type, as shown by the **globalType** element toward the end of the preceding XML schema. While you can have only one global element in an XML schema, you

can declare many elements (with different names) of the same global type in a single XML schema.

The following table shows the graphical representations of the different XML components in XQuery Mapper.

Table 2-5 Graphical Representation of XML Components

Component	Graphical Representation	Name in Preceding Example XML Schema
Global Element	 globalElement	globalElement
Local Element	 localElement	localElement
Global Type	 Global Types globalType	globalType
Attribute	 globalElement attribute	attribute defined for globalElement

Transforming Data Using XQuery Mapper

Examples: Data Transformation Using XQuery Mapper

The examples described here are based on the sample project that is included in the product. For information about opening the sample project, see [“Creating the XQuery Mapper Sample Project”](#) on page 2-2.

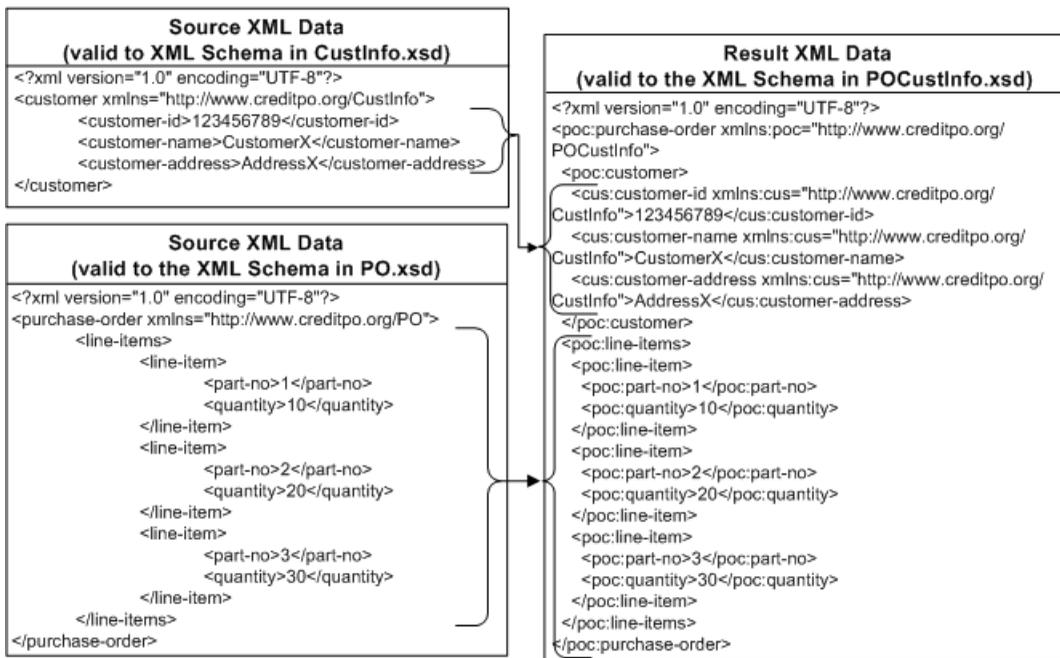
Examples are provided for the following scenarios:

- [Combining Data from Different Schemas](#)
- [Mapping Repeating Elements and Creating Joins](#)
- [Creating Unions](#)
- [Creating Repeating-Source to Nonrepeating-Target Transformations](#)
- [Creating Nonrepeating-Source to Repeating-Target Transformation](#)
- [Creating Nested If-Then-Else Expressions](#)
- [Creating FLWOR Expressions](#)
- [Using Recursive Schemas](#)
- [Grouping Data by Key Fields](#)

Combining Data from Different Schemas

You can use XQuery Mapper to combine content from two different schemas, as shown in the following figure.

Figure 3-1 Combining Data From Different Schemas



In this example, customer data (valid against `CustInfo.xsd`) is merged with a repeating element `line-items` (valid against `PO.xsd`) to form a single XML document that is valid against the `POCustInfo.xsd` schema.

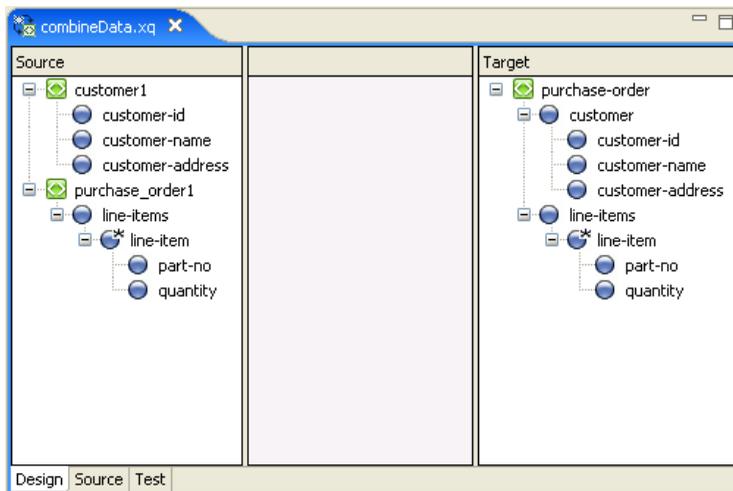
1. Launch Workshop and open the sample project.
 - For information about launching Workshop, see [“Launching XQuery Mapper” on page 2-2](#).
 - For information about opening the sample project, see [“Creating the XQuery Mapper Sample Project” on page 2-2](#).
2. Right-click the `xQuery Transformations` folder.

3. Choose **New > XQuery Transformation**.
4. Verify the name of the parent folder. For this example, the parent folder is **/XQuery Transformation/XQueryTransformations**.
5. Enter **combineData** as the file name and click **Next**.
6. Select the following source elements and click **Next**:
 - **CustInfo.xsd\customer**
 - **PO.xsd\purchase-order**
7. Select **POCustInfo.xsd\purchase-order** as the target element and click **Finish**.

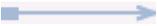
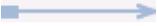
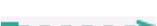
The **combineData.xq** file is created in the **/XQuery Transformation/XQueryTransformations** folder.

The source and target elements that you selected are displayed in the **Design** view, as shown in the following figure.

Figure 3-2 Design View of XQuery Transformation



8. Create links between the following source and target elements by dragging elements from the **Source** pane to the **Target** pane.

Source Element		Target Element
customer1		purchase-order\customer
customer1\customer-id		purchase-order\customer\customer-id
customer1\customer-name		purchase-order\customer\customer-name
customer1\customer-address		purchase-order\customer\customer-address
purchase_order1\line-items\line-item		purchase-order\line-items\line-item
purchase_order1\line-items\line-item\part-no		purchase-order\line-items\line-item\part-no
purchase_order1\line-items\line-item\quantity		purchase-order\line-items\line-item\quantity

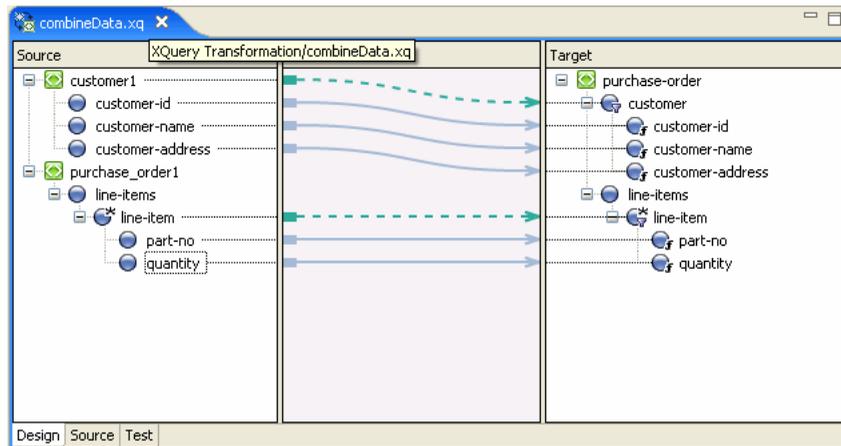
Note: Dotted lines represent **Structural** links, which are created between parent structures and do not map data directly.

Solid lines represent **Data** links, which convert the value of the source node directly to the value of the target node.

For more information, see [“Graphical Features in Design View” on page 2-27](#).

The links between the **Source** and **Target** elements are displayed, as shown in the following figure.

Figure 3-3 Data Transformation in Design View



9. Save the changes.

For information about testing XQuery transformations, see [“Testing Data Transformations”](#) on page 2-24.

Mapping Repeating Elements and Creating Joins

You can join data from XML files that are valid against different schemas (in this example, `PriceQuote.xsd`, `AvailableQuote.xsd`, and `taxrate.xsd`), and create an XML file that is valid against a single schema: `Quote.xsd`.

This example includes the following steps:

- [Step 1. Create an XQuery File](#)
- [Step 2. Add a Constraint](#)
- [Step 3. Add Data to Return Element](#)
- [Step 4. Add Function to Calculate Value of Quote](#)
- [Step 5. Add a Constraint with Multiple Conditions](#)
- [Test the XQuery](#)

Step 1. Create an XQuery File

In this step, we create an XQuery transformation by using the `AvailQuote.xsd`, `PriceQuote.xsd`, and `taxrate.xsd` schemas. Then, we map several `priceQuote` and `availRequest` source elements to corresponding target elements.

1. Launch Workshop and open the sample project.
 - For information about launching Workshop, see [“Launching XQuery Mapper” on page 2-2](#).
 - For information about opening the sample project, see [“Creating the XQuery Mapper Sample Project” on page 2-2](#).
2. Right-click the `xQuery Transformations` folder.
3. Choose **New > XQuery Transformation**.
4. Verify the name of the parent folder. For this example, the parent folder is `/xQuery Transformation/xQueryTransformations`.
5. Enter `Join` as the file name and click **Next**.
6. Select the following source elements and click **Next**:
 - `PriceQuote.xsd\priceQuote`
 - `PriceQuote.xsd\taxRate`
 - `AvailQuote.xsd\availRequest`
7. Select `Quote.xsd\quote` as the target element and click **Finish**.

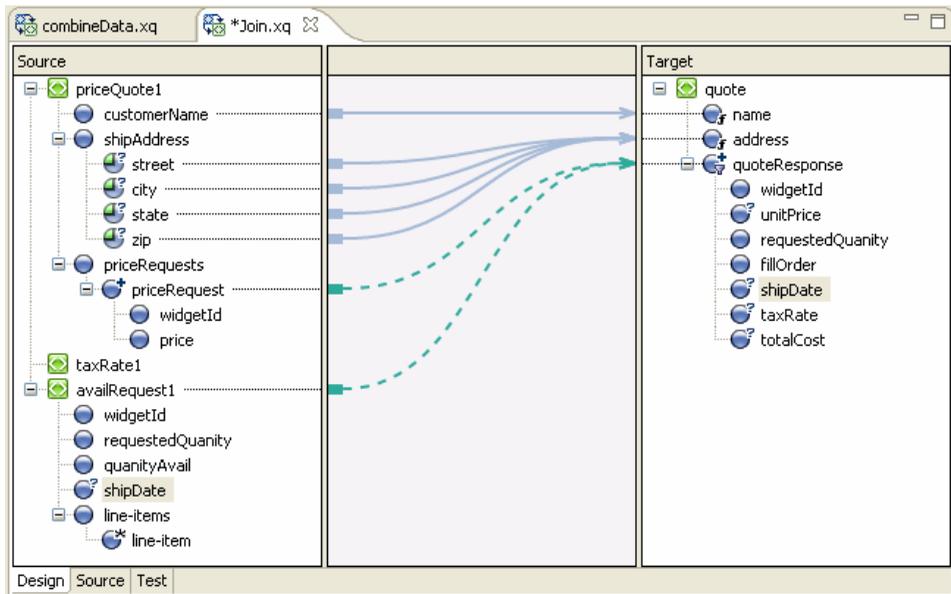
The `Join.xq` file is created in the `xQueryTransformation/xQueryTransformations` folder.
8. Create links between the following source and target elements by dragging elements from the **Source** pane to the **Target** pane.

Source Element	Target Element
<code>priceQuote1\customerName</code>	<code>quote\name</code>
<code>priceQuote1\shipAddress\street</code>	<code>quote\address</code>
<code>priceQuote1\shipAddress\city</code>	<code>quote\address</code>
<code>priceQuote1\shipAddress\state</code>	<code>quote\address</code>

Source Element	Target Element
<code>priceQuote1\shipAddress\zip</code>	<code>quote\address</code>
<code>priceQuote1\priceRequests\priceRequest</code>	<code>quote\quoteResponse</code>
<code>availRequest1</code>	<code>quote\quoteResponse</code>

The links are displayed, as shown in the following figure.

Figure 3-4 Data Transformation in Design View



9. Save the changes.

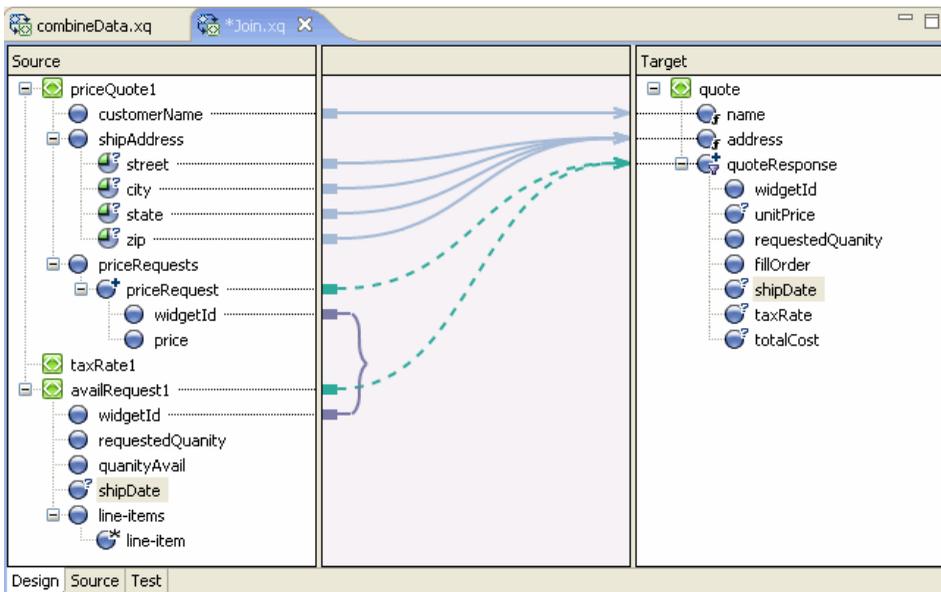
Step 2. Add a Constraint

The `priceQuote/priceRequests` and `availRequest` source elements share the common element `widgetId`. In this step, we add a constraint that if `widgetId` of the `availRequest` schema is equal to `widgetId` of the `priceQuote/priceRequests` element, then the query must return the target repeating element, `quoteResponse`.

1. Open `Join.xq` in the **Design** view.
2. Drag the `priceQuote1/priceRequests/priceRequest/widgetId` element from the **Source** pane and drop it on the `availRequest1/widgetId` element of the **Source** pane.

A connecting line appears between the two `widgetId` nodes in the **Source** pane, as shown in the following figure.

Figure 3-5 Adding a Constraint



3. Save the changes.
4. View the changes in the **Source** view.

The link between the two `widgetId` nodes is represented by a **where** clause within the **for** loop. The **where** clause specifies that the **for** loop must return the result of the expression only if the **where** clause is true. In this example, if `widgetId` of the `availRequest` element is equal to `widgetId` of the `priceRequest` element, the expression returns the XML data specified in the `quoteResponse` element.

Note: You can also view the **where** clause in the **Constraints** view.

The `quoteResponse` element is currently empty. We add content to the element in the next step.

Step 3. Add Data to Return Element

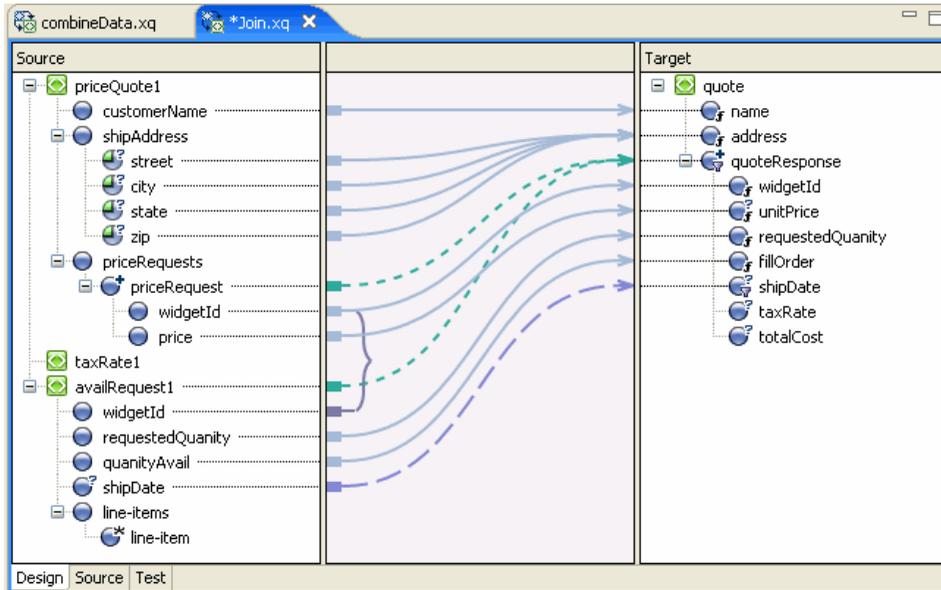
In this step, we add data links in the `quoteResponse` target element.

Open `Join.xq` in the **Design** view and create links between the following source and target elements:

Source Element	Target Element
<code>priceQuote1\priceRequests\priceRequest\widgetid</code>	<code>quote\quoteResponse\widgetid</code>
<code>priceQuote1\priceRequests\priceRequest\price</code>	<code>quote\quoteResponse\unitprice</code>
<code>availRequest1\requestedQuantity</code>	<code>quote\quoteResponse\requestedQuantity</code>
<code>availRequest1\quantityAvail</code>	<code>quote\quoteResponse\fillOrder</code>
<code>availRequest1\shipDate</code>	<code>quote\quoteResponse\shipDate</code>

The links between the **Source** and **Target** elements are displayed, as shown in the following figure.

Figure 3-6 Adding Data in the QuoteResponse Element



Step 4. Add Function to Calculate Value of Quote

In this step, we add a function to calculate the total value of the quote.

1. Open the `Join.xq` file in **Source** view.
2. Insert the following function declaration at any point in the source code between the namespace declarations and the `Join` function call. You can, for example, insert it just before the `Join` function declaration.

Listing 3-1 calculateTotalPrice Function

```
declare function xf:calculateTotalPrice(  
    $taxRate as xs:float,  
    $quantity as xs:float,  
    $price as xs:float)  
  
as xs:float {  
  
    let $taxQuantity := ($taxRate * $quantity)  
    let $totalTax := ($taxQuantity * $price)
```

```

let $costNoTax := ($quantity * $price)
let $totalCost := ($totalTax + $costNoTax)

return $totalCost
};

```

3. Switch to the **Design** view.

Note: `Join.xq` now includes two function declarations: `calculateTotalPrice` and `Join`. When more than one function exists in an XQuery file, the function with the same name as the XQ file is rendered in the **Design** view. In this case, the `Join` function is displayed in the **Design** view.

4. In the **Target** pane, select the `totalCost` node. Keep it selected for the next step.
5. Select the **Target Expression** view and select the **General** option.
6. Insert the following code in the **General Expression** pane.

```

xf:calculateTotalPrice($taxRate1,$availRequest/ns1:requestedQuantity,$pr
iceRequest/ns0:price)

```

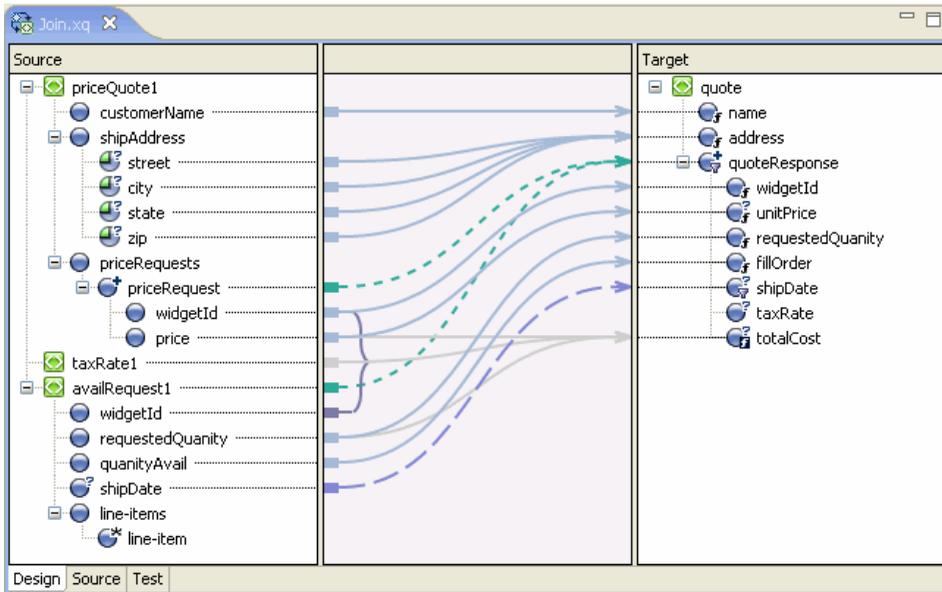
7. Click **Apply**.

The expression is added to the `totalCost` element in the XQuery.

8. Save the changes.

The **Design** view shows the calculation for the `totalCost` target element.

Figure 3-7 totalCost Calculation in Design View



Step 5. Add a Constraint with Multiple Conditions

You can create a constraint by using the **Where Clause** pane of the **Constraints** view to limit the target repeating elements that are returned by the XQuery. At run time, the **for** loop in the XQuery iterates over only those repeating elements that satisfy the **where** clause.

In this step, we add another condition (resulting in a complex condition) to the **where** clause of the **for** loop to further limit the data returned by the **for** loop.

1. Open the `join.xq` file.
2. In the **Design** view, select the link between the `availRequest1` source element and the `quote\quoteResponse` target element.

The single condition that makes up the **where** clause is displayed in **Where Clause** pane of the **Constraints** view.

```
data($priceRequest/ns0:widgetId) = data($availRequest/ns1:widgetId)
```

3. Drag the `availRequest1/requestedQuantity` element from the **Source** pane and drop it in the **Left Hand Expression** area of the **Where Clause** pane.

The left hand expression of the where clause is created as follows:

```
data($availRequest/ns1:requestedQuantity)
```

4. Select the < operator.
5. Remove the text in the **Right Hand Expression** area of the **Where Clause** pane, and enter "50".

Note: Enter the number 50 within quotation marks ("50", not 50).

6. From the **Join Type** field select the **AND** option.

The **Join Type** determines how the conditions that make up the **where** clause are evaluated at run time.

7. Click **Add**. The second condition is added to the **where** clause of the **for** loop.
8. Save the changes.

This step completes the creation of the following **where** clause.

Listing 3-2 Where Clause

```
where (data($availRequest/ns1:widgetId) = data($priceRequest/ns0:widgetId)
and data($availRequest/ns1:requestedQuantity) < "50")
```

Test the XQuery

Perform the following steps to verify that the XQuery works when both the conditions of the **where** clause you created are satisfied.

1. Switch to the **Test** view.
2. In the **Source Data** pane of the **Test** view, select **priceQuote** in the **Source Variable** field, and click the **Generate Data** icon.
3. Note the value of the **widgetId** element in the test XML data.

```
<ns0:widgetId>value</ns0:widgetId>
```
4. In the **Source Data** pane, select **availRequest** in the **Source Variable** field, and click the **Generate Data** icon.

5. Edit the value the value of the `widgetId` element in the test XML data to match the value displayed in the `priceQuote` test XML data.
`<ns0:widgetId>value</ns0:widgetId>`
6. Locate the `requestedQuantity` element and edit the value to a number less than 50.
For example: `<ns0:requestedQuantity>25</ns0:requestedQuantity>`
7. In the **Result Data** pane, click the **Test XQuery** icon and view the results of the XQuery.

Creating Unions

In this example, we use the **Union** option in the **Constraints** view to construct an XQuery that maps data of the same type into larger sets of data.

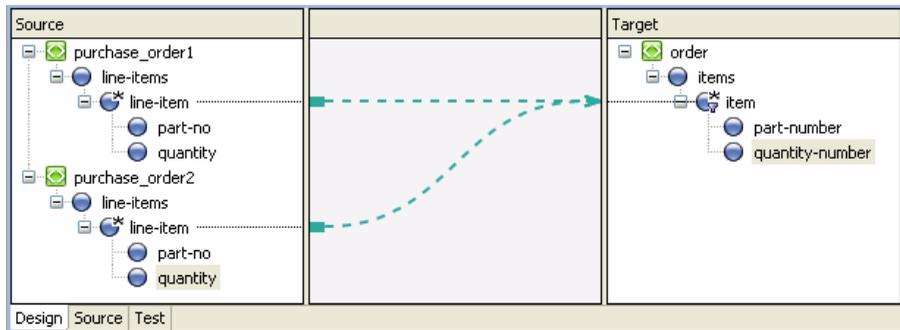
1. Launch Workshop and open the sample project.
 - For information about launching Workshop, see [“Launching XQuery Mapper” on page 2-2](#).
 - For information about opening the sample project, see [“Creating the XQuery Mapper Sample Project” on page 2-2](#).
2. Right-click the **xQuery Transformations** folder.
3. Choose **New > XQuery Transformation**.
4. Verify the name of the parent folder. For this example, the parent folder is `/xQuery Transformation/xQueryTransformations`.
5. Enter **union** as the file name and click **Next**.
6. In the **Source Types** dialog box, select `PO.xsd\purchase-order` twice, and then click **Next**.
Note: To add an element more than once, you must change the parameter name.
7. Select `Order.xsd\order` as the target type and then click **Finish**.
The `union.xq` file is created.

8. Create links between the following source and target elements:

Source Element	Target Element
<code>purchase-order1\line-items\line-item</code>	<code>order\items\item</code>
<code>purchase-order2\line-items\line-item</code>	<code>order\items\item</code>

The following figure shows how the links appear in the **Design** view.

Figure 3-8 Creating a Union



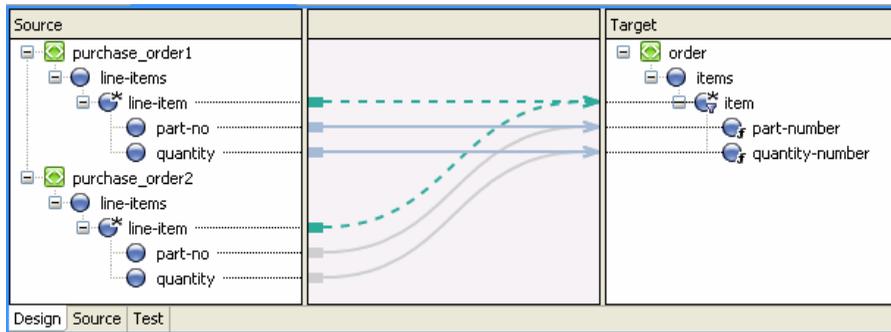
9. Select the link between the `$purchase-order1/line-items/line-item` source element and the `order/items/item` target element.
10. In the **Constraint Type** pane of the **Constraints** view, select **Union**.
11. Create links between the following source and target elements:

Source Element	Target Element
<code>purchase-order1\line-items\part-no</code>	<code>order\items\item\part-number</code>
<code>purchase-order1\line-items\quantity</code>	<code>order\items\item\quantity-number</code>

Note: When you want to create links between source and target elements of the same name, you can use the **Induce Map** option instead of creating the links manually. For more information, see [“Right-Click Menu Options” on page 2-27](#).

Since the two structural links have a union constraint, a set of implied data links between the second set of subelements is generated as shown in [Figure 3-9](#). The gray lines represent implied links that were created because **Union** was selected as the constraint type.

Figure 3-9 Creating Implied Links



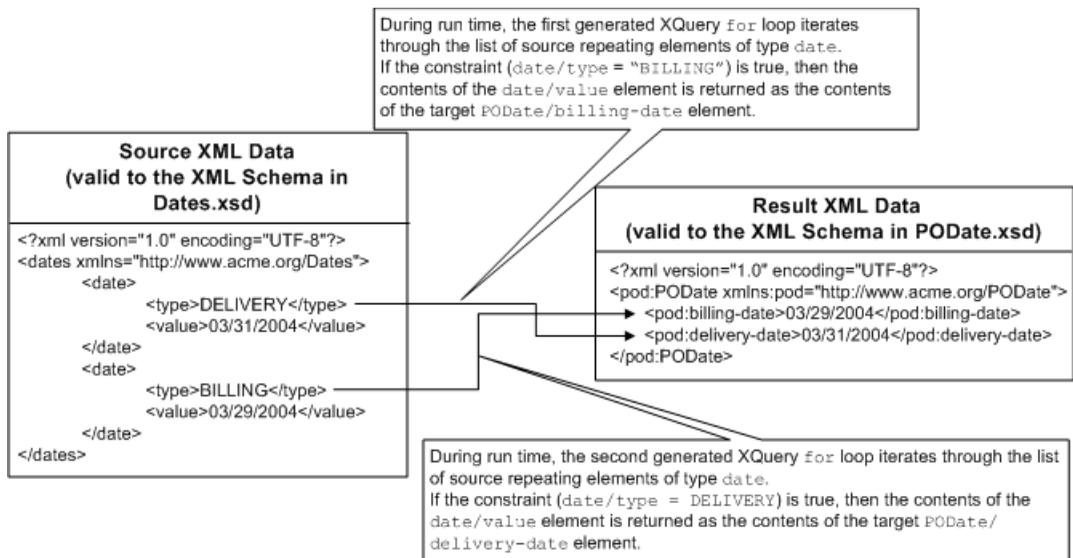
12. Save the changes.

For information about testing XQuery transformations, see [“Testing Data Transformations”](#) on page 2-24.

Creating Repeating-Source to Nonrepeating-Target Transformations

In this example, we map a repeating source XML element to a nonrepeating target XML element. The following figure depicts the transformations that we create in this example.

Figure 3-10 Repeating-Source-to-Nonrepeating Target Transformation

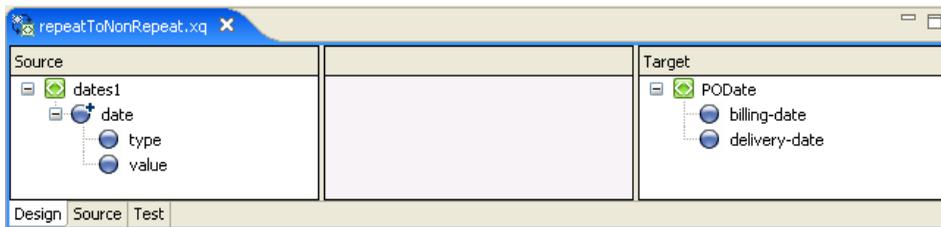


1. Launch Workshop and open the sample project.
 - For information about launching Workshop, see [“Launching XQuery Mapper” on page 2-2.](#)
 - For information about opening the sample project, see [“Creating the XQuery Mapper Sample Project” on page 2-2.](#)
2. Right-click the **xQuery Transformations** folder.
3. Choose **New > XQuery Transformation**.
4. Verify the name of the parent folder. For this example, the parent folder is **/xQuery Transformation/xQueryTransformations**.
5. Enter **repeatToNonRepeat** as the file name and click **Next**.

6. Select `Dates.xsd\dates` as the source schema and click **Next**.
7. Select `PODate.xsd\PODate` as the target schema and click **Finish**.

The `repeatToNonRepeat.xq` file is created and displayed, as shown in the following figure.

Figure 3-11 Repeating-Source-to-Nonrepeating-Target Data Transformation



8. Create a link between the `dates1/date` repeating element in the **Source** pane and the `PODate/billing-date` element in the **Target** pane.
Keep this link selected for the next step.
9. Select the **Constraints** view.
10. Drag the `dates1/date/type` element from the **Source** pane to the **Left Hand Expression** area of the **Where Clause** pane in the **Constraints** view.
11. Select the = operator.
12. In the **Right Hand Expression** area, enter `"BILLING"` (including the quotation marks), and click **Add**.
13. Create a link between the `dates1/date/value` element in the **Source** pane and the `PODate/billing-date` element in the **Target** pane.
The constraint that you created in the preceding steps specifies that the value of the `dates1/date/type` element in an XML document must be compared with the value `"BILLING"`.
At run time, if the value of the `dates1/date/type` element is `"BILLING"`, the XQuery returns the value of `dates1/date/value` as the value of `billing-date`.
14. Create a link between the `dates1/date` repeating element in the **Source** pane and the `PODate/delivery-date` element in the **Target** pane.
Keep this link selected for the next step.

15. Drag the `dates1/date/type` element from the **Source** pane to the **Left Hand Expression** area of the **Where Clause** pane in the **Constraints** view.
16. Select the = operator.
17. In the **Right Hand Expression** area, enter **"DELIVERY"** (including the quotation marks), and click **Add**.
18. Create a link between the `dates1/date/value` repeating element in the **Source** pane and `PODate/delivery-date` element in the **Target** pane.

The constraint created in the preceding steps specifies that the value of the `dates1/date/type` element in an XML document must be compared to the value "DELIVERY".

At run time, if the value of the `dates1/date/type` element is "DELIVERY", the XQuery returns the value of `dates1/date/value` as the value of `delivery-date`.

Figure 3-12 Repeating-Source-to-Nonrepeating-Target Data Transformation



19. Save the changes.
- For information about testing XQuery files, see [“Testing Data Transformations” on page 2-24](#).

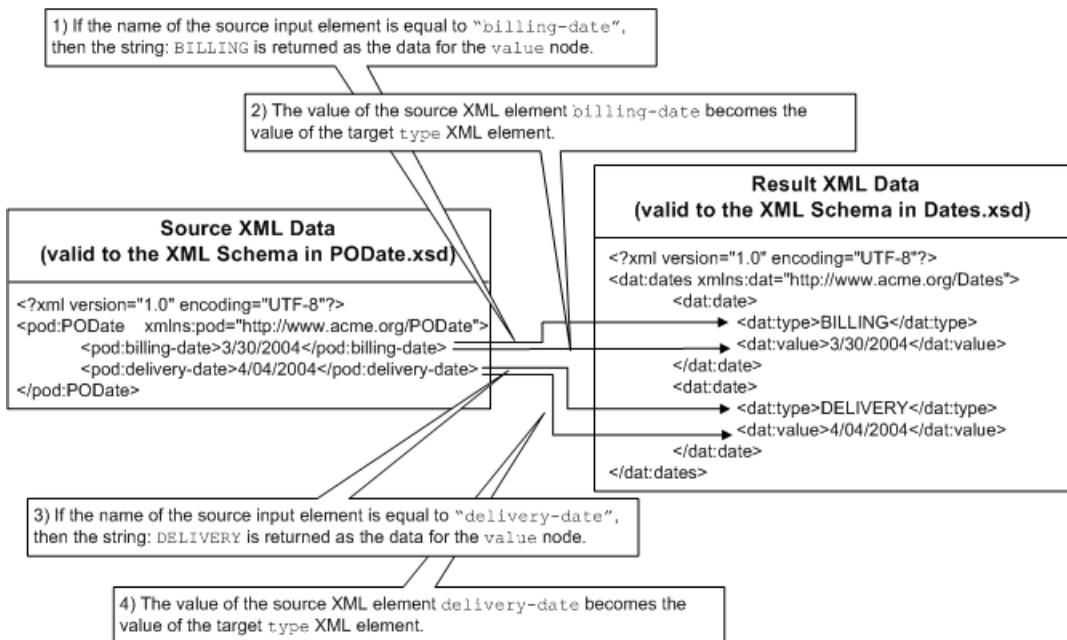
Creating Nonrepeating-Source to Repeating-Target Transformation

In this example, we map a nonrepeating source element to a repeating target element.

The following figure depicts the transformations that we create in this example.

Figure 3-13 Nonrepeating-Source-to-Repeating-Target Transformation

During run time, the query executes the following actions:



1. Launch Workshop and open the sample project.
 - For information about launching Workshop, see [“Launching XQuery Mapper” on page 2-2](#).
 - For information about opening the sample project, see [“Creating the XQuery Mapper Sample Project” on page 2-2](#).
2. Right-click the **xQuery Transformations** folder.

3. Choose **New > XQuery Transformation**.
4. Verify the name of the parent folder. For this example, the parent folder is **/XQuery Transformation/XQueryTransformations**.
5. Enter **nonRepeatToRepeat** as the file name and click **Next**.
6. Select **PODate.xsd\PODate** as the source schema and click **Next**.
7. Select **Dates.xsd\dates** as the target schema and click **Finish**.
8. The **nonRepeatToRepeat.xq** file is created and displayed in the **Design** view.
9. Create links between the following source and target elements:

Source Pane	Target Pane
<code>pODate1/billing-date</code>	<code>dates/date</code>
<code>pODate1/delivery-date</code>	<code>dates/date</code>

The following listing shows the XQuery code that is generated.

Listing 3-3 XQuery Code

```
<ns1:dates>
{
  for $PODate in $PODate1/ns0:billing-date union $PODate1/ns0:delivery-date
  return
    <ns1:date/>
}
</ns1:dates>
```

At run time, the **for** loop is executed twice. In the first execution, the iteration variable **\$PODate** is equal to the first element in the union **\$PODate1/ns0:billing-date**; in the second execution, **\$PODate** is equal to **\$PODate1/ns0:delivery-date**.

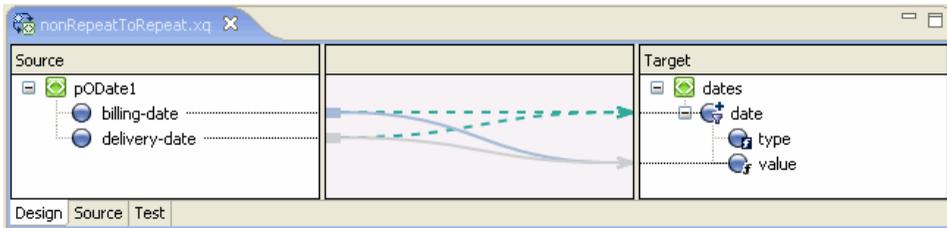
The XQuery returns two empty XML elements with the tag `<ns1:date/>`.

In the following steps, we add the XQuery code to return the billing and delivery dates to the query.

10. Switch to the **Design** view.
11. Create a link between the `pODate1/billing-date` source element and the `dates/value` target element in the **Target** pane.

Two data links are created, as shown in the following figure.

Figure 3-14 Creating a Union for Structural Links



The following structural links were joined when we created the link from `pODate1/billing-date` to `dates/value`.

- `pODate1/billing-date` to `dates/date`
- `pODate1/delivery-date` to `dates/date`

A second data link between the `pODate1/delivery-date` element and `dates/value` element was created automatically.

12. Create a link between the `pODate1/billing-date` source element and the `dates/type` target element.

Two data links are created.

Keep the `pODate1/billing-date` to `dates/type` link selected for the next step.

13. Select the **Target Expression** view.
14. Select the **If Then Else** option.

The following XQuery if-then-else expression is added to the link:

```
if (fn:boolean("true")) then
    data($PODate)
```

```
else
  ()
```

15. In this step, we add a condition to the **if** section of the if-then-else expression.

a. Select **If Condition** in the **Expression Structure** area.

The **Edit If Condition** pane is displayed.

b. In the **Expression Functions** view, expand **Node Functions**.

c. Drag the `local-name` function to the **Left Hand Expression** area of the **Edit If Condition** pane. Leave the `$node-var` argument in the function selected.

d. Select the **Expression Variables** view.

e. Drag the `PODate` structural link variable to the `$node-var` argument of the `local-name` function in the **Left Hand Expression** area of the **Edit If Condition** pane.

f. Select the = operator.

g. In the **Right Hand Expression** area of the **Edit If Condition** pane, enter `"billing-date"` (including quotation marks), and then click **Add**.

The following condition is added to the **if** section of the if-then-else expression:

```
fn:local-name($PODate)="billing-date"
```

16. Select **Then Expression** in the **Expression Structure** area.

The **Edit Then Condition** pane is displayed.

17. Replace the existing text with `"BILLING"` (including quotation marks), and then click the **Apply** icon.

18. Select **Else Expression** in the **Expression Structure** area.

The **Edit Else Condition** pane is displayed.

19. Replace the existing text with `"DELIVERY"` (including quotation marks), and then click the **Apply** icon.

20. Select **If Then Else** in the **Expression Structure** area.

The following XQuery code is displayed in the **Expression Structure** pane.

```
if (fn:local-name($PODate) = "billing-date") then
  "BILLING"
```

```
else
  "DELIVERY"
```

21. Save the changes.

For information about testing XQuery files, see [“Testing Data Transformations” on page 2-24](#).

Creating Nested If-Then-Else Expressions

In this example, we create an XQuery transformation that calculates price based on a widget ID and tax rate for a state. We create an if-then-else expression for the following logic:

- If widget ID is from 0 to 200, the price is \$10.00
- Else if the widget ID is from 201 to 400, price is \$20.00
- Else If the widget ID is from 401 to 600, price is \$30.00

This example includes the following steps:

- [Step 1. Create the XQuery Transformation](#)
- [Step 2. Create the First “If” Condition](#)
- [Step 3. Create the First Nested If-Then-Else Condition](#)
- [Step 4. Create the Second Nested If-Then-Else Condition](#)

Step 1. Create the XQuery Transformation

In this step, we create an XQuery transformation by using the `PurchaseAgree.xsd` and `Supplier.xsd` schemas.

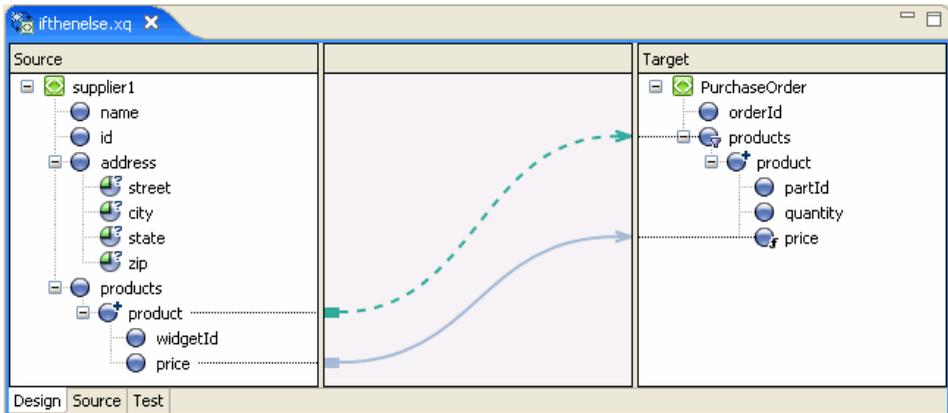
1. Launch Workshop and open the sample project.
 - For information about launching Workshop, see [“Launching XQuery Mapper” on page 2-2](#).
 - For information about opening the sample project, see [“Creating the XQuery Mapper Sample Project” on page 2-2](#).
2. Right-click the `xQuery Transformations` folder.
3. Choose **New > XQuery Transformation**.

4. Verify the name of the parent folder. For this example, the parent folder is `/XQuery Transformation/XQueryTransformations`.
5. Enter `ifthenelse` as the file name and click **Next**.
6. Select `Supplier.xsd\Supplier` as the source type and click **Next**.
7. Select `PurchaseAgree.xsd\PurchaseOrder` as the target type and click **Finish**.
The `ifThenElse.xq` file is created.
8. Create links between the following source and target elements.

Source Element	Target Element
<code>supplier1/products/product</code>	<code>PurchaseOrder/products/product</code>
<code>supplier1/products/product/price</code>	<code>PurchaseOrder/products/product/price</code>

The links are displayed, as shown in the following figure.

Figure 3-15 XQuery Transformation for If-Then-Else Example



Step 2. Create the First “If” Condition

In this step, we create an `if` expression to specify that if the widget ID is from 0 to 200, then the price is \$10.00.

1. Select the link between the `supplier1/products/product/price` source element and the `PurchaseOrder/products/product/price` target element.
2. Select the **Target Expression** view.
3. Select the **If Then Else** expression type.
4. In the **Expression Structure** pane, select **If Condition**.
The **Edit If Condition** pane is displayed.
5. Drag the `supplier1/products/product/widgetID` element from the **Source** pane and drop it in the **Left Hand Expression** pane.
6. Select the `>=` operator.
7. In the **Right Hand Expression** pane, enter `"0"` (including quotation marks), and click **Add**.
8. Select the `<=` operator.
9. In the **Right Hand Expression** pane, enter `"200"`.
10. Click the arrow next to the **Update** button and select **Add**.
11. In the **Expression Structure** pane, select **Then Condition**.
The **Edit Then Condition** pane is displayed.
12. In the **Edit Then Expression** pane, delete the existing data and enter `"$10.00"`.
13. Click the **Apply** icon.
14. In the **Expression Structure** pane, select **If Then Else**.
The if-then expression is displayed as shown in the following listing.

Listing 3-4 If-Then Expression

```
if ((xs:string(data($product/ns0:widgetId)) >= "0"
    and xs:string(data($product/ns0:widgetId)) <= "200")) then
    "$10.00"
else
    ()
```

Step 3. Create the First Nested If-Then-Else Condition

In this step, we create an `if` expression to specify that if the widget ID is from 201 to 400, then the price is \$20.00. To accomplish this, we insert a nested if-then-else inside the `else` expression we created in the previous step.

1. In the **Expression Structure** pane, right-click **Else Expression**, and select **Insert Nested If-Then-Else**.
2. In the nested **If-Then-Else** expression, select **If Condition**.
3. From the **Source** pane, drag the `widgetID` element and drop it in the **Left Hand Expression** pane.
4. Select the `>=` operator.
5. In the **Right Hand Expression** pane, enter `"201"` (including quotation marks), and click **Add**.
6. Select the `<=` operator.
7. In the **Right Hand Expression** pane, enter `"400"`.
8. Click the arrow next to the **Update** button and select **Add**.
9. In the **Expression Structure** pane, select **Then Expression**.
10. In the **Edit Then Expression** pane, enter `"$20.00"`.
11. Click the **Apply** icon.
12. In the **Expression Structure** pane, select **If Then Else**.

The if-then-else expression appears as shown in the following listing.

Listing 3-5 Nested If-Then-Else Expression

```
if ((xs:string(data($product/ns0:widgetId)) >= "0"
    and xs:string(data($product/ns0:widgetId)) <= "200")) then
    "$10.00"
else
    if ((xs:string(data($product/ns0:widgetId)) >= "201"
        and xs:string(data($product/ns0:widgetId)) <= "400")) then
```

```
"$20.00"  
else  
  ()
```

Step 4. Create the Second Nested If-Then-Else Condition

In this step, we create an `if` expression to specify that if the widget ID is from 401 to 600, then the price is \$30.00. To accomplish this, we insert a nested if-then-else expression within the `else` expression that we created in the previous step.

1. In the **Expression Structure** pane, select the `else` clause of the nested if-then-else expression created in [“Step 3. Create the First Nested If-Then-Else Condition” on page 3-27](#)
2. Right-click and select **Insert Nested If-Then-Else**.
3. Select the `if` condition in the nested if-then-else expression that we just created.
4. From the **Source** pane, drag `widgetID` and drop it in the **Left Hand Expression** pane.
5. Select the `>=` operator.
6. In the **Right Hand Expression** pane, enter `"401"`, and then click **Add**.
7. Select the `<=` operator.
8. In the **Right Hand Expression** pane, enter `"600"`, and then click **Add**.
9. In the **Expression Structure** pane, select **Then Expression**.
10. In the **Edit Then Expression** pane, enter `"$30.00"`, and click the **Apply** icon.
11. In the **Expression Structure** pane, select **If Then Else**.

The nested if-then-else expression is as shown in the following listing.

Listing 3-6 Nested If-Then-Else Expression

```
if ((xs:string(data($product/ns0:widgetId)) >= "0"  
  and xs:string(data($product/ns0:widgetId)) <= "200")) then  
  "$10.00"
```

```

else
  if ((xs:string(data($product/ns0:widgetId)) >= "201"
    and xs:string(data($product/ns0:widgetId)) <= "400")) then
    "$20.00"
  else
    if ((xs:string(data($product/ns0:widgetId)) >= "401"
      and xs:string(data($product/ns0:widgetId)) <= "600")) then
        "$30.00"
    else
      ()

```

For information about testing XQuery files, see [“Testing Data Transformations” on page 2-24](#).

Creating FLWOR Expressions

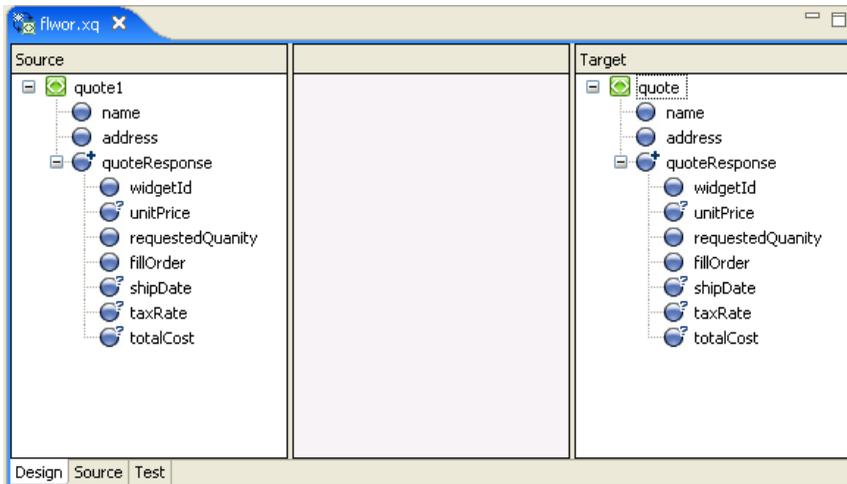
In this example, we use a For-Let-Where-Order By-Return expression to extract widget IDs from a quotation, for items with a total value more than 2000.

1. Create the XQuery file.
 - a. Launch Workshop and open the sample project.
 - For information about launching Workshop, see [“Launching XQuery Mapper” on page 2-2](#).
 - For information about opening the sample project, see [“Creating the XQuery Mapper Sample Project” on page 2-2](#).
 - b. Right-click the **xQuery Transformations** folder.
 - c. Choose **New > XQuery Transformation**.
 - d. Verify the name of the parent folder.
 - e. Enter **flwor** as the file name and click **Next**.
 - f. Select **quote.xsd\quote** as the source type and click **Next**.

- g. Select `Quote.xsd\quote` as the target type and click **Finish**.

The `flwor.xq` file is created, as shown in the following figure.

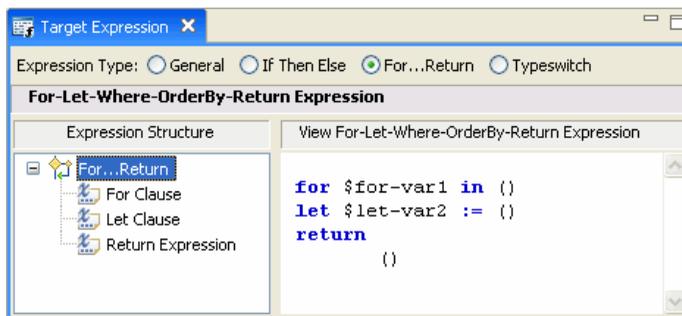
Figure 3-16 XQuery Example - FLWOR Expression



- 2. Create the FLWOR expression.
 - a. Click on the `quote\quoteResponse` repeating element in the target pane.
 - b. In the **Target Expression** view, select the **For...Return** option.

The view changes, as shown in the following figure.

Figure 3-17 FLWOR Expression



- c. In the **Expression Structure** pane, select **For Clause**.

The **Edit For Clause** pane is displayed.

- d. In the **Variable** field, replace the existing value with `quote`.
 - e. In the **Expression Variables** view, expand the `quote1` node.
 - f. Drag `quote1/quoteresponse` from the **Expression Variables** view, and drop it in the **Single Expression** field.
 - g. Click **Update**.
3. Design the **Let** clause.
- Note:** For this example, the let clause is not essential. It is used here merely to illustrate how to design it in XQuery Mapper.
- a. In the **Expression Structure** pane, select **Let Clause**.
The **Edit Let Clause** pane is displayed.
 - b. In the **Variable** field, replace the existing value with `widget`.
 - c. Select the **Expression Variables** view.
 - d. Expand the `quote` node within the **Structural Link** folder.
 - e. Drag `quote/widgetID` from the **Expression Variables** view and drop it in the **Single Expression** field.
 - f. Click **Update**.
4. Design the **Where** clause.
- a. In the **Expression Structure** pane, right-click **For...Return** and select **Insert Where Clause**.
 - b. Select **Where Clause**.
The **Edit Where Condition** pane is displayed.
 - c. Drag `quote/totalCost` from the **Structural Link** folder of the **Expression Variables** view, and drop it in the **Left Hand Expression** field.
 - d. Select the `>` operator.
 - e. Enter `2000` in the **Right Hand Expression** area.
 - f. Click **Add**.

5. Design the **Order By** clause.
 - a. In the **Expression Structure** pane, right-click **For...Return**, and select **Insert Order By Clause**.
 - b. Select **Order By Clause**.

The **Edit Order By Clause** pane is displayed.
 - c. In the **Sort Order** field, select **ascending**.
 - d. In the **Single Expression** field, enter `$widget`, which is the name of the variable that is declared in the `let` clause.
 - e. Click **Update**.
6. Design the **Return** expression.
 - a. In the **Expression Structure** pane, select **Return Expression**.
 - b. In the **Expression Variables** view, expand the **Structural Link** folder.
 - c. Drag `quote` from the **Expression Variables** view, and drop it in the **Single Expression** field.
 - d. Click the **Apply** icon.
7. Save the changes.

You can view the source code of the FLWOR expression by selecting **For...Return** in the **Expression Structure** pane. The code is as shown in the following listing.

Listing 3-7 Code for FLWOR Expression

```
for $quote in ($quote1/quoteResponse)
let $widget := ($quote/widgetId)
where $quote/totalCost > 2000
order by $widget ascending
return
    $quote
```

For information about testing XQuery transformations, see [“Testing Data Transformations” on page 2-24](#).

Using Recursive Schemas

In this example, we create a data transformation with schemas that have recursive elements.

An element in a schema is considered recursive when it contains a child element of the same type as the parent, as shown in the example in [Listing 3-8](#). In this example, the `product` element is a recursive element because it is of type `productType`, and `productType` contains a `child-product` element which is also of type `productType` (`productType` refers to itself).

Listing 3-8 Example of Recursive Schema

```
<?xml version="1.0"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
  targetNamespace="http://www.acme.org/Product"
  xmlns="http://www.acme.org/Product" elementFormDefault="qualified"
  attributeFormDefault="unqualified">
  <xs:complexType name="productType">
    <xs:sequence>
      <xs:element name="part-description" minOccurs="0"
        maxOccurs="unbounded" type="xs:string" />
      <xs:element name="child-product" minOccurs="0"
        maxOccurs="unbounded" type="productType" />
    </xs:sequence>
  </xs:complexType>
  <xs:element name="product" type="productType">
  </xs:element>
</xs:schema>
```

Perform the following steps to create a transformation with recursive schemas:

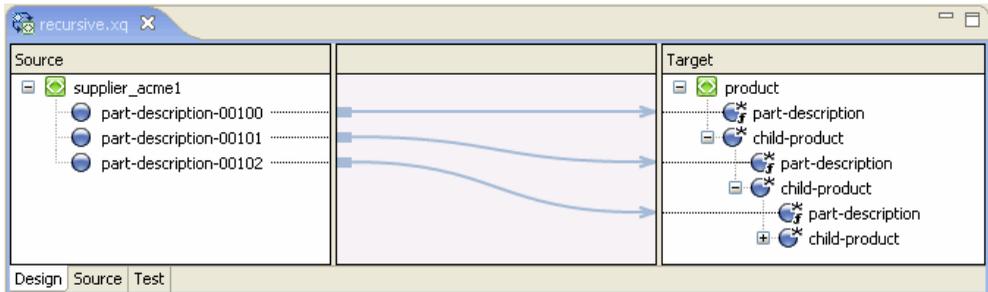
1. Launch Workshop and open the sample project.
 - For information about launching Workshop, see [“Launching XQuery Mapper” on page 2-2](#).
 - For information about opening the sample project, see [“Creating the XQuery Mapper Sample Project” on page 2-2](#).
2. Right-click the **xQuery Transformations** folder.
3. Choose **New > XQuery Transformation**.
4. Verify the name of the parent folder. For this example, the parent folder is **/xQuery Transformation/xQueryTransformations**.
5. Enter **recursive** as the file name and click **Next**.
6. Select **SupplierAcme.xsd\supplier_acme** as the source schema and click **Next**.
7. Select **Product.xsd\product** as the target schema and click **Finish**.

The **recursive.xq** file is created.
8. Create links between the following source and target elements:

Source Element	Target Element
<code>supplier_acme1\part-description-00100</code>	<code>product\part-description</code>
<code>supplier_acme1\part-description-00101</code>	<code>product\child-product\part-description</code>
<code>supplier_acme1\part-description-00101</code>	<code>product\child-product\child-product\part-description</code>

The following figure shows the links from the source elements to the recursive **child-product** target elements.

Figure 3-18 Mapping Recursive Elements



9. Save the changes.

For information about testing XQuery files, see [“Testing Data Transformations” on page 2-24](#).

Grouping Data by Key Fields

You can use the **Group by Key Fields** feature to group data based on one or more key values.

Note: The Group-By feature is not supported graphically in XQuery Mapper and there is no representation of the XQuery in the **Design** view. You must write the Group-By expression in the **Source** view.

The following listing shows the XML document that we use as input in this example.

Listing 3-9 Example Input XML Document

```
<input-warehouse-inventory xmlns="http://www.creditpo.org/repkeyin">
  <input-line-item>
    <input-warehouse-id>Warehouse1</input-warehouse-id>
    <input-location-desc>Location1</input-location-desc>
    <input-part-no>1</input-part-no>
    <input-quantity>10</input-quantity>
  </input-line-item>
  <input-line-item>
    <input-warehouse-id>Warehouse2</input-warehouse-id>
```

Examples: Data Transformation Using XQuery Mapper

```
<input-location-desc>Location2</input-location-desc>
<input-part-no>2</input-part-no>
<input-quantity>20</input-quantity>
</input-line-item>
<input-line-item>
  <input-warehouse-id>Warehouse1</input-warehouse-id>
  <input-location-desc>Location1</input-location-desc>
  <input-part-no>3</input-part-no>
  <input-quantity>30</input-quantity>
</input-line-item>
</input-warehouse-inventory>
```

In this example, we use the `input-warehouse-id` and `input-location-desc` elements as the key fields to group data in the output document:

The first and third instances of the `input-line-item` repeating element contain the same values for the `input-warehouse-id` and `input-location-desc` elements: `Warehouse1` and `Location1` respectively.

The goal of this example is to write an XQuery that groups the first and third instances of the line items, by using the `Warehouse1` and `Location1` keys in the output document, as shown in [Listing 3-10](#).

Listing 3-10 Example Output XML Document

```
<ns0:output-inventory xmlns:ns0="http://www.creditpo.org/repkeyout" ;>
  <ns0:output-warehouse-inventory>
    <ns0:output-warehouse-id>Warehouse1</ns0:output-warehouse-id>
    <ns0:output-location-desc>Location1</ns0:output-location-desc>
    <ns0:output-line-item>
      <ns0:output-part-no>1</ns0:output-part-no>
```

```

        <ns0:output-quantity>10</ns0:output-quantity>
    </ns0:output-line-item>
    <ns0:output-line-item>
        <ns0:output-part-no>3</ns0:output-part-no>
        <ns0:output-quantity>30</ns0:output-quantity>
    </ns0:output-line-item>
</ns0:output-warehouse-inventory>
<ns0:output-warehouse-inventory>
    <ns0:output-warehouse-id>Warehouse2</ns0:output-warehouse-id>
    <ns0:output-location-desc>Location2</ns0:output-location-desc>
    <ns0:output-line-item>
        <ns0:output-part-no>2</ns0:output-part-no>
        <ns0:output-quantity>20</ns0:output-quantity>
    </ns0:output-line-item>
</ns0:output-warehouse-inventory>
</ns0:output-inventory>

```

Perform the following steps to create a Group-By expression:

1. Launch Workshop and open the sample project.
 - For information about launching Workshop, see [“Launching XQuery Mapper” on page 2-2](#).
 - For information about opening the sample project, see [“Creating the XQuery Mapper Sample Project” on page 2-2](#).
2. Right-click the `xquery Transformations` folder.
3. Choose **New > XQuery Transformation**.
4. Verify the name of the parent folder. For this example, the parent folder is `/XQuery Transformation/XQueryTransformations`.

5. Enter `groupby` as the file name and click **Next**.
6. Select `regroupKeyFldIn.xsd\input-warehouse-inventory` as the source schema, and click **Next**.
7. Select `regroupKeyFldOut.xsd\output-inventory` as the target schema, and click **Finish**.

The `groupby.xq` file is created.

8. Select the **Source** view.
9. Replace the existing code with the code in the following listing.

Listing 3-11 XQuery Code for Group-By Expression

```
declare namespace ns0 = "http://www.creditpo.org/repkeyin";
declare namespace ns1 = "http://www.creditpo.org/repkeyout";
declare function Regrouping($input-warehouse-inventory as
  element(ns0:input-warehouse-inventory))
  as element(ns1:output-inventory) {
  <ns1:output-inventory>
  {
    for $input-line-item in $input-warehouse-inventory/ns0:input-line-item
    group $input-line-item as $group by
      $input-line-item/ns0:input-warehouse-id as $key0,
      $input-line-item/ns0:input-location-desc as $key1
    return
      <ns1:output-warehouse-inventory>
        <ns1:output-warehouse-id>{ data($key0) }</ns1:output-warehouse-id>
        <ns1:output-location-desc>{ data($key1) }</ns1:output-location-desc>
        {
          for $group0 in $group return
            <ns1:output-line-item>
              <ns1:output-part-no>{xs:byte(data($group0/ns0:input-part-no))}
              </ns1:output-part-no>
        }
  }
}
```

```

        <ns1:output-quantity>{xs:byte
          (data($group0/ns0:input-quantity)) }
      </ns1:output-quantity>
    </ns1:output-line-item>
  }
</ns1:output-warehouse-inventory>
}
</ns1:output-inventory>
};

declare variable $input-warehouse-inventory as
element(ns0:input-warehouse-inventory) external;

Regrouping($input-warehouse-inventory)

```

10. Save the changes.

The changes are not visible in the **Design** view.

11. With the `groupby.xq` file open in the **Source** view, select the **Test** view.

12. In the **Source Data** pane, click the **Import** icon.

13. Import the `Regrouping.xml` file provided in the sample project (from the `XML Transformation/XML/` folder).

14. In the **Result Data** pane, select the **Test XQuery** icon.

The result of the XQuery is displayed, as shown in [Listing 3-10](#).

Examples: Data Transformation Using XQuery Mapper

Upgrading XQuery Code

This section describes the procedure for upgrading inline XQuery code and stand alone XQuery files from XQuery 2002 to 2004.

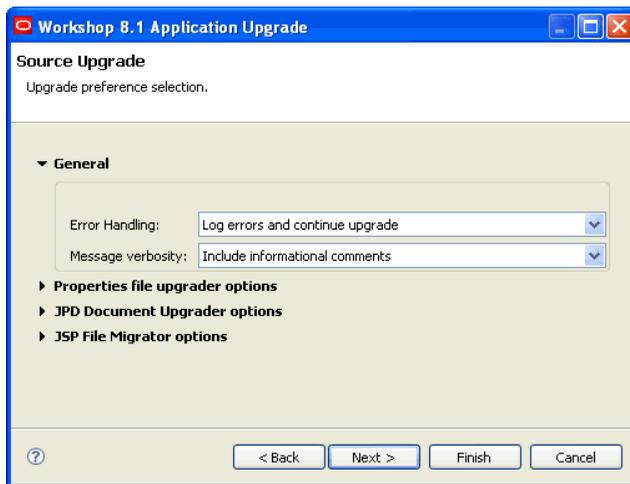
This section contains the following topics:

- [Upgrading Inline XQuery Code](#)
- [Upgrading XQuery Files](#)

Upgrading Inline XQuery Code

When you upgrade a WLI 8.x project to 10g Release 3 (10.1.3), by default, inline XQuery 2002 code (embedded in JPD files) is not converted to XQuery 2004. You can convert the inline XQuery code to XQuery 2004, by performing the following steps in the **Source Upgrade** screen of the upgrade wizard.

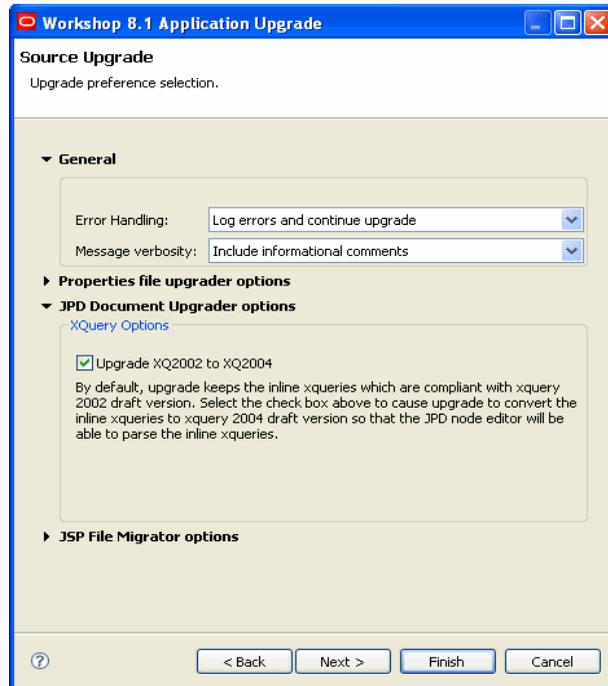
Figure A-1 Source Upgrade Screen of the Upgrade Wizard



1. Expand **JPD Document Upgrader options**.

2. Select the **Upgrade XQ2002 to XQ2004** check box, as shown in the following figure.

Figure A-2 Upgrade XQ2002 to XQ2004



3. Continue with the application upgrade process.

When the upgrade process is completed, XQuery 2002-compliant code that is embedded in JPD files (**inline** XQuery code) is converted to XQuery 2004.

Upgrading XQuery Files

XQuery 2002-compliant **XQuery files** (unlike **inline XQuery code**) are not converted automatically to XQuery 2004 during the application upgrade process.

You can upgrade XQuery 2002-compliant XQuery files to XQuery 2004 by right-clicking the XQuery file in the **Navigator** view in Workshop and choosing **Upgrade to XQuery 2004**.

However, the following XQuery constructs and functions are not converted to XQuery 2004:

- Constructs for which the syntax has changed in XQuery 2004:

Table A-1 Constructs For Which the Syntax Has Changed in XQuery 2004

XQuery 2002	XQuery 2004	Note
precedes	<<	In some cases, merely changing the syntax might not give correct results.
follows	>>	
add-timezone-to-dateTime	fn:adjust-dateTime-to-timezone	The syntax and parameter type have changed. In the XQuery 2004 syntax, the operation is addition regardless of the sign of the parameter. In XQuery 2004, the operation depends on the sign of the parameter.
remove-timezone-from-dateTime	fn:adjust-dateTime-to-timezone	
document()	fn:doc()	The parameter type and return type might be different.
foreach	for \$i in ...	There is no direct replacement for foreach . The construct must be rewritten by using a for loop.
some \$x in (1, 2, 3), xs:integer \$y in (2, 3, 4)	some \$x in (1,2,3), \$y in (2,3,4)	Existence of the optional type declaration makes it difficult to reliably convert this construct.
every \$x in (1, 2, 3), xs:integer \$y in (2, 3, 4)	every \$x in (1,2,3), \$y in (2,3,4)	

- Functions for which the semantics have changed in XQuery 2004:

Table A-2 Functions For Which Parameter Types and Order Have Changed in XQuery 2004

XQuery 2002	XQuery 2004
<code>xf:resolve-uri(anyURI \$base, anyURI \$relative)</code>	<code>fn:resolve-uri(\$relative as xs:string?, \$base as xs:string)</code>
<code>xf:distinct-values(item* \$srcval => item*)</code>	<code>fn:distinct-values (\$arg as xdt:anyAtomicType*) as xdt:anyAtomicType*</code>

- Constructs for which the casting rule has changed: **eq**, **ne**, **lt**, **gt**, **le**, and **ge**.
- Functions for which the return type has changed in XQuery 2004
 - Most time/date-related functions and operators
 - Queries containing **sortby**
 - **generic-divide**
 - **div**
 - **expanded-QName**
 - **namespaces**
 - **QName**
 - **node-name**
 - **namespace-uri**
 - **subsequence**
 - **integer-divide**
 - **round**
 - **name**
- Function not supported in XQuery 2004: **julianDay-from-date**

During the upgrade process, XQuery 2002 constructs and functions that are not supported by XQuery 2004 are flagged with comment entries indicated by “**: warning :**” in the code. In addition, a warning message is displayed during the upgrade process, informing you that unsupported constructs and functions were encountered.

Upgrading XQuery Code

After the upgrade process, you can identify the constructs and functions that are not upgraded to XQuery 2004 by looking for the “**: warning :**” comments. You must then manually change the unsupported constructs and functions to make them comply with the XQuery 2004 standard.